Sketch Recognition User Interfaces: Guidelines for Design and Development

Christine Alvarado
MIT CSAIL
Cambridge, MA 02139
calvarad@csail.mit.edu

Abstract

We present a free-sketch recognition-based tool for creating Microsoft Power Point diagrams. Unlike many previous pen-based interfaces, this tool performs aggressive and robust recognition, allowing the user to sketch freely while the system recognizes the sketched diagram and seamlessly imports it into Power Point. Although pen-based user interfaces have been developed and studied, little has been said about the user interface issues involved in developing sketch *recognition* user interfaces. We present initial user interface guidelines for creating sketch recognition user interfaces (SkRUIs) based on informal, iterative user studies of our Power Point tool. Finally, based on our experience, we claim that a number of iterative design techniques developed for traditional user interfaces cannot be readily applied to SkRUIs. We discuss which techniques are best suited to the design and development of SkRUIs.

Introduction

In recent years there have been a number of developments in pen and sketch-based interfaces (Landay & Myers 1995; Newman *et al.* 2003; Saund *et al.* 2003). To date, however, sketch recognition has not been reliable enough to use extensively in these interfaces. To be usable, most current pen-based computer design tools explicitly avoid recognizing users' strokes, or aim to recognize only a restricted set of symbols, often those drawn with a single stroke. We aim to make sketch recognition reliable enough that it can be incorporated into sketch-based early stage design tools for a wide range of domains. Users will be able to interact with these tools by sketching their designs freely and naturally, without having to use special gestures or commands to enable the system to understand their drawings. We call this emerging class of interfaces *Sketch Recognition User Interfaces* or *SkRUIs* to emphasize that they are not merely pen-based interfaces, but also interfaces that perform free-sketch recognition.

SkRUIs represent an important new style of interaction that has been explored to only a limited extent in previous work. Work by Mankoff *et. al*, for example, explores some human computer interaction (HCI) aspects of recognition-based systems by providing tools to handle ambiguity in a wide range of recognition-based interfaces, including some limited sketch recognition interfaces (Mankoff, Hudson, & Abowd 2000). However, because of the difficulty of sketch recognition, to date most pen-based research has focused either on HCI or on sketch recognition technology, but not both. Work in the first category has explored the user interface challenges of building pen-based systems that perform only limited or no recognition (Newman *et al.* 2003; Anderson *et al.* 2004; Saund *et al.* 2003). Work in the second category has focused on building robust recognition, but addresses the user experience to only a limited extent (Alvarado & Davis 2001b; Hammond & Davis 2002; Lank, Thorley, & Chen 2000; Mahoney & Fromherz 2002).

This paper addresses both HCI and sketch recognition by exploring the user interface aspects of a recognition-based diagram creation tool that robustly recognizes naturally drawn diagrams and automatically imports them into Microsoft Power Point. Our tool combines the ease of drawing on paper with Power Point's sophisticated presentation capabilities.

Robust recognition within this tool is achieved through the use of a multi-domain sketch engine called SketchREAD, presented in previous work (Alvarado & Davis 2004). SketchREAD allows users to draw freely in a number of domains and robustly interprets the user's strokes as they are drawn. Although not ready to be used in SkRUIs for complex domains, SketchREAD may currently be used in domains where the shapes have little overlap and are drawn spatially separated, as in the Power Point diagrams we consider here.

We explore a number of user interface challenges in incorporating free sketch recognition into a design tool, including when to display recognition feedback, how to integrate sketching with editing, and how the user should correct recognition errors. We present guidelines for creating this type of SkRUI based on a series of informal user studies with a prototype implementation. These guidelines take into consideration both the requirements needed to maintain robust sketch recognition and what felt natural to users.

Finally, we evaluate the utility of a number of iterative design and evaluation techniques—including paper prototyping, heuristic evaluation, computer prototyping, and qualitative user testing on the complete system—for the development of sketch recognition interfaces. We found that the interactive nature of sketching made paper prototyping and final system evaluation more informative than heuristic eval-

Appeared in *AAAI 2004 Symposium on Making Pen-Based Interaction Intelligent and Natural, 2004.* pp.8-14.

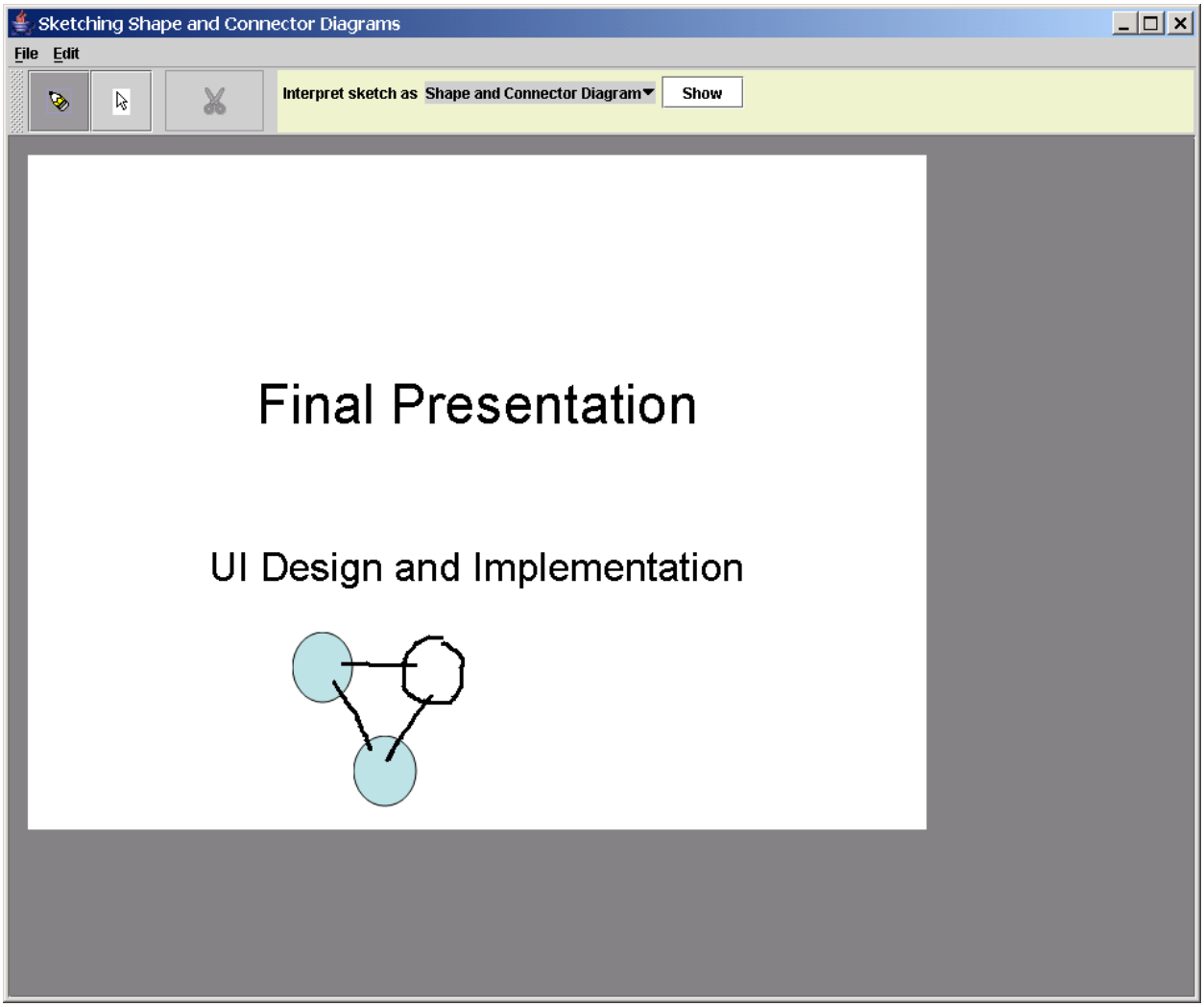


Figure 1: The final design of our Power Point diagram creation tool.

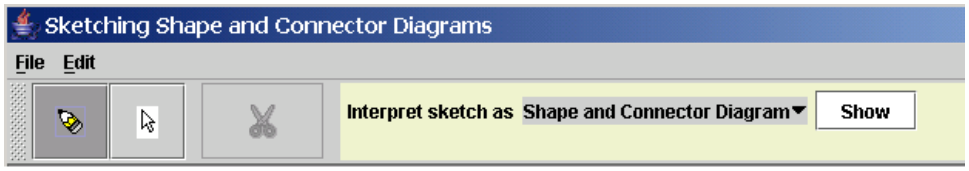
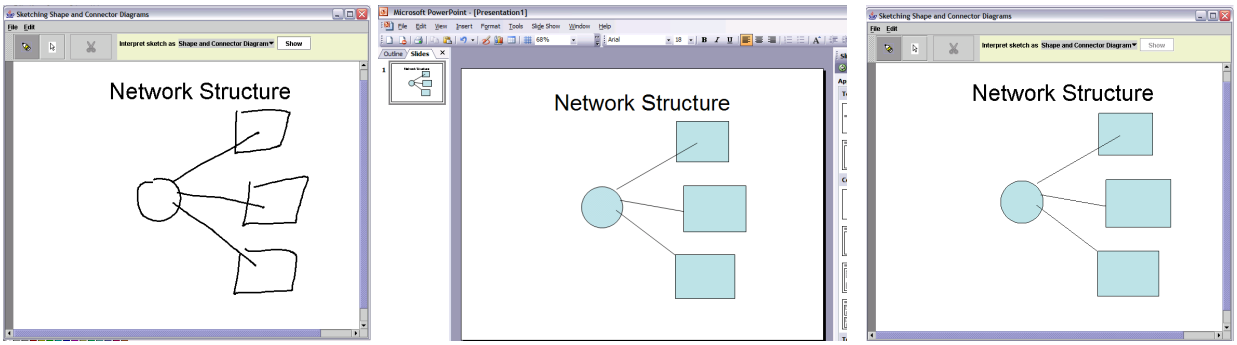


Figure 2: A close-up of the top of the Power Point diagram creation tool.



(a) A sketched diagram

(b) The sketched objects are automatically imported into Power Point

(c) The recognized objects replace the user's strokes

Figure 3: An example illustrating the interaction between the Power Point window and the diagram creation tool.

uation and computer prototyping. We believe the results of our exploration will help guide the design process of future SkRUIs.

Application

We constructed sketch recognition-based Power Point digram creation tool in order to explore the user interface challenges in creating SkRUIs. With this tool users may create diagrams consisting of shapes (ellipses and quadrilaterals) and connectors (lines and arrows) by sketching them on a Tablet PC. The system recognizes the different shapes as they are drawn naturally—the user does not have to perform any additional action to indicate which shape is being drawn. Users can draw the strokes in any order and may use any number of strokes for each shape. Sketch recognition is performed using an engine described in other work (Alvarado & Davis 2004).

The interface design is shown in Figures 1 and 2. Our sketch recognition application communicates with Power Point, but runs in a separate window. The user sketches diagrams directly onto a reproduction of the slide; these sketches are recognized and automatically imported into Power Point.

Our goal was to make the interaction between our diagram creation tool and Power Point as seamless as possible. One way in which we accomplished this goal was to ensure that the diagram creation window and Power Point application are automatically synchronized to contain the same information without requiring any explicit action from the user. When switches to the Power Point window (or clicks the "Show" button), the recognized objects are added to the Power Point slide (Figure 3(b)). When the user switches back to the diagram creation tool, the recognized objects appear on the slide in place of the user's strokes (Figure 3(c)). (Note that although the user's strokes are recognized as they are drawn, recognition feedback is not given until after the objects are added to Power Point.) The diagram tool also automatically updates its content in response to changes made within Power Point. As one example, the slide showing in the diagram creation system is automatically updated to reflect the current slide visible in the sketch recognition window.

Our tool also provides a number of editing capabilities including the ability to move and delete portions of the diagram. Our system performs free-sketch recognition, attempting to recognize all of the user's strokes as pieces of a diagram. Developing pen-based editing commands that would not be mistaken for sketched strokes was not trivial. One way that we accomplished this goal was to include an explicit editing mode, which the user enters by clicking the arrow toggle button at the top of the window (Figures 1 and 2). In edit mode, the user may select a single item by putting her pen on it or select one or more items by dragging a box around them (Figure 4). A selected item appears highlighted (Figure 5). While in edit mode, the user cannot draw new items. To resume drawing the user must tap the pencil button at the top to return to sketch mode.

Standard edit mode makes it easy for the system to distinguish between sketched strokes and editing gestures, but we found that users often forgot to return to sketch mode and would try to draw while still in edit mode. We introduced *online edit mode* to make the interaction between editing and sketching more intuitive. To allow the user to edit while still in sketch mode, we provide a selection gesture that is not easily confused with sketched strokes. To select an item using online edit mode, the user holds down the pen until the cursor changes from a pencil to an arrow. If the user is on top of an item, that item becomes selected. If the user is not on top of an item, the pen can be used to drag out a selection box to select multiple items. When the user lifts her pen, the system returns to sketch mode and the user may sketch new items. However, her selected items remain highlighted, indicating that these items may be moved or deleted using the same pen movements and commands as in edit mode. Pen strokes that begin on top of the selected item will be interpreted as editing gestures rather than sketched strokes.

In our initial interviews, users expressed a desire to add unrecognized annotations to their diagrams. In response, our system supports both recognized drawing, where the user's strokes become clean Power Point objects, and unrecognized drawing, where the user's strokes appear on the slide exactly as they were drawn. Automatically distinguishing

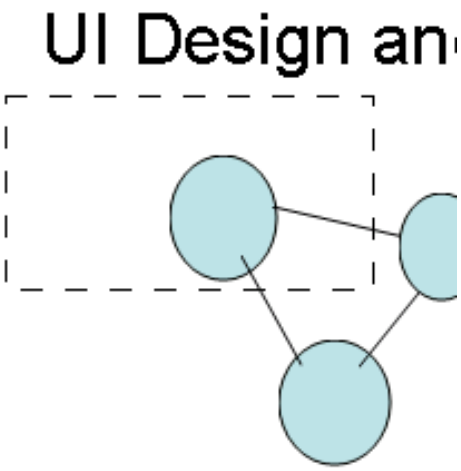


Figure 4: The user drags a selection box

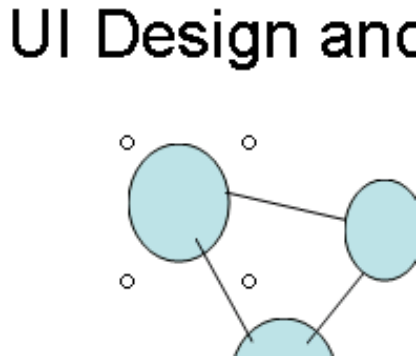


Figure 5: A selection is made

between recognized diagrams and unrecognized annotations is currently an unsolved problem for a free-sketch recognition system. Instead, the user can explicitly switch between recognized and unrecognized drawing using the combo box at the top of the window (Figure 2).

The features that are not included in our diagram creation tool, including copy and paste, alignment, and adding text to the diagram, can be performed by the user in Power Point. We discuss below how distributing these features between the diagram creation tool and Power Point affected the user's experience.

System Evaluation

Throughout the design process, we used a number of formative, or early-stage, evaluation techniques to guide the system's design. The recognition technology to support SkRUIs is still emerging, and consequently these interfaces have not been previously studied from a UI perspective. Of course, our eventual goal is to build SkRUI-based tools that provide a quantitative improvement over existing tools. However, to enable the construction of powerful tools, we still need a better qualitative understanding of what users want from these interfaces. Accordingly, here our goal was to understand users' perceptions of our tool and what they wanted from such a tool, rather than to prove that we had created an effective system.

We used several evaluation methods throughout the design process. Our design/evaluation stages included a paper prototype tested with users, a low-fidelity computer prototype both tested with users and evaluated using heuristics, and the final system tested with users.

Each user-focused evaluation method involved three users recruited from within the MIT graduate student community. All users had recently worked on creating presentations, although one was not very familiar with Power Point. Our participants reflected the target community for this tool in that graduate students often create presentations for teaching and seminars. The participant who was unfamiliar with Power Point allowed us to understand what aspects of our sketch recognition system were challenging for users unfamiliar with the complete system, not just the diagram creation interface.

In each test, users were read a high-level introduction to the diagram creation tool that briefly described what the system could and could not do, but did not describe how to use it. Following this briefing, each user was asked to perform three prototypical diagram creation tasks using our diagram creation tool. Each task was selected to explore a specific aspect of interacting with the system. The first task explored how the user used the tool to create a new drawing, beginning with the task of creating a new slide (which is done in Power Point). The second task explored the interaction between the pen-based creation tool and keyboard text entry. In this task we asked users to create a diagram and then label it using Power Point. Our system does not support handwriting recognition, and we wanted to see if this presented a problem or if users felt comfortable using the keyboard to enter text. The third task explored the interaction between sketching and editing using the pen. We asked the user to sketch a diagram and then move and delete pieces of the diagram and then continue to draw.

Design Guidelines

Our design/evaluation process gave us insights into what users wanted from sketch recognition user interfaces. In this section we give guidelines and advice for incorporating sketch recognition into diagram creation tools based on this design/evaluation process and our knowledge of the requirements for robust sketch recognition.

Display recognition results only when the user is done sketching. The system should display recognition results after a user has completed sketching, rather than after every stroke. Previous work has suggested that recognition feedback can distract the user during the design task (Hong *et al.* 2002); we have observed this effect in previous work (Alvarado & Davis 2001a). In response, our diagram creation interface did not display recognition feedback while users sketched. We found that users sketched their diagrams with-

out pausing between shapes and did not appear to expect recognition feedback. Only one user paused after drawing his first stroke, expecting that stroke to be recognized, and he acknowledged that his expectations were based on his interaction with our previous recognition interface.

We stated above that the system should wait until the user is done sketching to display recognition results. Unfortunately, determining when a user is done sketching is a difficult problem. Implicit cues such as how long the user has paused since her last stroke are not always reliable. Particularly in design tasks, users may pause in the middle of their design, but do not want to receive recognition feedback at this point.

In determining when a user is done sketching, our observations suggest that a SkRUI should rely on explicit cues. In our evaluation, we found window focus was a reliable indicator that the user had completed a sketch: When users had completed a sketch, they usually switched back to the Power Point window. Users also clicked the "Show" button at the top of the window (Figure 2). Neither method of indicating they were done drawing seemed to inconvenience the user because they could perform these actions at their own convenience. We conclude that asking the user to explicitly inform the system when a diagram is complete is a viable option for a SkRUI.

Finally, although we claim the system should wait until the user is done sketching to provide recognition feedback, it is acceptable to give feedback even if the user is only temporarily done sketching. In some cases, users stopped sketching, switched to the Power Point window (to add text to their diagrams, for example) and then returned to the diagram creation tool to continue sketching. In these cases it was acceptable that recognition had occurred before they were completely done with the diagram because they were not actively sketching when the recognition was performed. Users did not seem to mind adding sketched strokes on top of recognized diagrams.

Provide obvious indications to distinguish free sketching from recognition. Allowing the user to draw strokes that remain unrecognized by the system (e.g., an informal, sketched annotation next to a clean recognized diagram) presents a challenge for a SkRUI. Because SketchREAD recognizes freely drawn sketches, where symbols can be composed of more than one stroke, it is difficult to determine when the user wants her strokes recognized and when she wants them to remain unrecognized. To support this functionality, we provide explicit "free sketch" and "recognition" modes. The combo box at the top of the window (Figure 2, currently set to "Shape and Connector diagram") indicates the system's current recognition mode.

We found that users could understand the difference between the modes, but many users forgot what mode they were in, despite the label at the top of the window. One user drew an entire digram in free-sketch mode and could not figure out why the system failed to recognize his strokes. The system needs to use more obvious clues (such as changing the cursor or the stroke style, perhaps) to help users remain aware of what recognition mode they are in.

Restrict recognition to a single domain until automatic domain detection becomes feasible. Although our application supports the recognition of only one domain—relationship diagrams—we would eventually like to build a tool that supports the recognition of more than one domain. We experimented with such a system in our early paper prototypes, asking users to draw both relationship diagrams and circuit diagrams. In this prototype, user's had to explicitly choose the recognition domain before creating a diagram. This explicit choice was necessary because the paper prototype simulated our actual recognition engine, which needs to know in which domain the user is drawing. Automatic domain detection represents a technical challenge that our system cannot yet handle, so we asked users to explicitly choose the domain using a combo box at the top of the diagram creation window.

Despite this request, users tended to attribute universal recognition capabilities to the system and did not understand that they had to specify a particular domain. For example, we asked users to draw a relationship diagram immediately after they had drawn a circuit diagram. If they failed to change the recognition domain, we indicated that their sketch had been incorrectly recognized as a (messy) circuit diagram. None of our three users performing this task correctly switched domains, and none of them could figure out what had gone wrong. We conclude that single-domain tools will be most effective until automatic domain detection becomes a feasible alternative, or until we explore more effective ways to help the user understand the system's underlying recognition model.

Incorporate pen-based editing. The system should incorporate as much editing capability as possible (including copy, paste, alignment, resizing, etc.) into a sketch-based diagram creation interface because users wanted to simultaneously create and clean up their drawings. In our final testing, one user repeatedly redrew a circle until it was the exact same size as another on the screen, then carefully aligned it with two she had already drawn. The ability to copy, paste, and align her objects could have saved a considerable amount of effort. Although this behavior may not be as prevalent for rough design tasks, based on previous work (Alvarado & Davis 2001a; Adler & Davis 2004), we anticipate that users will still want the ability to copy and paste parts of their drawings.

Sketching and editing should use distinct pen motions. The system should support pen-based editing gestures that are distinct from any pen movement the user would make while sketching. These gestures allow the user to avoid explicitly specifying sketching and editing mode, but should be specifically designed designed so that they are not easily confused with inking strokes in a free-sketch system.

SkRUIs require large buttons. When given the option of buttons, menus, and keyboard shortcuts, users chose to use the button with few exceptions. We found that these buttons should be larger than traditional buttons because of the difficulties in targeting with the pen: In our initial tests, users could not reliably click buttons that were the same size as

those designed for a mouse-based interface. When we enlarged the button-size they had no trouble.

The pen must always respond in real time. We are trying to build a system that is as natural as sketching on paper. To accomplish this goal, the system must not allow the computation demands of recognition to interfere with the user's drawing. It is acceptable that *recognition* not occur in exactly real time; users tolerated a slight recognition delay once they were done drawing. However, it is extremely important that the demands of recognition not overwhelm the system's ability to display the user's strokes exactly when they are drawn. In our tests, when the user's strokes were delayed, users became agitated, repeatedly drawing the same shape until it appeared under their pen as they drew. These added strokes confused the recognition system, causing it to add extra shapes to the user's diagrams. Because the users thought they had drawn a single shape, they could not understand the existence of the extra shapes in the diagram.

As recognition technology improves, the potential for building more powerful SkRUIs will increase. We believe these initial guidelines will help researchers build usable and intuitive systems that make use of this new class of interface.

Iterative Design and Evaluation Techniques

Iterative UI design involves user evaluation early in the design process when designs are relatively easy to change. Iterative design is widely supported as an effective approach to UI development, and a number of standard iterative evaluation techniques have been developed including paper prototyping, heuristic evaluation, low-fidelity computer prototyping, and Wizard of Oz techniques. However, these techniques were developed for traditional button, keyboard and menu-based Graphical UIs (GUIs) or speech recognition interfaces, and we found that many of them are not well suited to the development of SkRUIs. We briefly introduce the techniques mentioned above, and then we discuss the characteristics of SkRUIs that make the application of these traditional techniques difficult and present suggestions as to which techniques may best be used for SkRUI development.

The goal of each iterative evaluation is to determine how well a user can understand and use an interface and to uncover the confusing or unusable aspects of the interface. This evaluation can (and should) be done at all stages of a system's implementation. Paper prototyping is used to test an interface before any implementation takes place. The user interacts with a paper version of an interface, while a human controller manipulates the paper to allow the user to understand how the computer system would respond. Low-fidelity computer prototyping is used to evaluate a more polished interface without having to construct the entire back-end. In this type of testing, the system produces canned responses to a few commonly performed user tasks. Finally, informal, qualitative user testing on the complete system can be performed to uncover interface problems before quantitative studies are run. At each of these stages, the interface can also be evaluated without users by examining it according to a number of established heuristics (Nielsen).

There are two main differences between SkRUIs and traditional GUIs that affect the applicability of various design techniques. First, SkRUIs are generally not command-based interfaces. Sketching is a creative process, and the user does not think of her strokes as commands to the system. This characteristic differentiates SkRUIs not only from GUIs, but also from most speech recognition interfaces, in which the user speaks in order to issue commands to the system. Second, unlike in a GUI, the freedom of sketching relative to button clicks or menus makes it difficult to reliably anticipate the user's actions.

Based on these observations, we suggest that any prototype evaluation tool must match the user's expectations of the system's reactivity. Paper prototyping is useful, because the user does not expect a real-time reaction to her strokes and is content with an explanation about what the computer's reaction will be in the implemented system. On the other hand, we found that any reactive computer prototype must have a fully implemented recognition back end, because when the user works with a computer, she expects immediate and exact reactions to her strokes, even if she is informed that the system is a prototype with limited functionality.

Heuristic evaluation also had limited utility in our tests because it did not involve an kind of interaction between the user and the system. While it was useful to catch "standard" UI problems such as the lack of sufficient help and documentation, also could not evaluate the user's understanding of the system's response because it is based on an analysis of a static version of the interface.

Wizard of Oz techniques, in which a human secretly acts as the computer by interpreting the user's strokes and generating the proper response, would be useful for the iterative design of SkRUIs, but they will be difficult to construct due to unpredictability of user input. Wizard of Oz techniques rely on the "wizard's" ability to quickly produce a system response for the user's input. In a speech recognition system, this task is challenging but has been effective ((Dahlback, Jonsson, & Ahrenberg 1993)). A Wizard of Oz system for SkRUI design faces the added challenge of handling two-dimensional input (e.g., the wizard cannot simply respond with a recognized symbol, he also must place the symbol in the appropriate location on the screen).

We believe that a powerful Wizard of Oz framework for SkRUIs, while useful, may not be ready in time for the development of the initial SkRUIs. Due to the difficulties of creating low-fidelity computer prototypes, we suggest that, rather than rather than spending time creating a computer prototype with canned responses to the user's actions, the designer should focus on implementing a strong recognition back end with a front end that can be easily modified.

Future Work and Conclusion

The diagram creation tool presented here is an early prototype, not yet ready for deployment or formal user studies. We aim to extend its functionality and conduct a more formal comparison between this system and Power Point to help us better understand the utility of our system. A more formal evaluation will also help us refine the design and evaluation guidelines presented in this paper.

In conclusion, we have discussed a new class of user interface that is not only pen-based, but free-sketch recognition-based. As recognition technology improves, the potential for building SkRUIs will increase. SkRUIs are, of course, fundamentally user interfaces, and it is cruicial always to keep in mind the user's perspective when building any UI. We presented an early-stage SkRUI developed with an explicit focus on the user's perspective. Based on iterative design of this interface we gave guidelines for the development of SkRUIs and discussed the strengths and weaknesses of existing iterative evaluation techniques for the development of future SkRUIs. We believe that the work presented here is an important first step for developing sketch recognition interfaces that are both intelligent and usable.

References

Adler, A., and Davis, R. 2004. Speech and sketching for multimodal design. In *Proceedings of the 9th International Conference on Intelligent User Interfaces*, 214–216. ACM Press.

Alvarado, C., and Davis, R. 2001a. Preserving the freedom of sketching to create a natural computer-based sketch tool. In *Human Computer Interaction International Proceedings*.

Alvarado, C., and Davis, R. 2001b. Resolving ambiguities to create a natural sketch based interface. In *Proceedings. of IJCAI-2001*.

Alvarado, C., and Davis, R. 2004. Sketchread: A sketch recognition engine for any domain. In *Proc. of UIST '04*.

Anderson, R.; Anderson, R.; Simon, B.; Wolfman, S.; VanDeGrift, T.; and Yasuhara, K. 2004. Experiences with a tablet pc based lecture presentation system in computer science courses. In *Proc. SIGCSE '04*.

Dahlback, N.; Jonsson, A.; and Ahrenberg, L. 1993. Wizard of oz studies - why and how. *Intelligent User Interfaces (IUI93)* 193–200.

Hammond, T., and Davis, R. 2002. Tahuti: A geometrical sketch recognition system for uml class diagrams. *AAAI Spring Symposium on Sketch Understanding* 59–68.

Hong, J.; Landay, J.; Long, A. C.; and Mankoff, J. 2002. Sketch recognizers from the end-user's, the designer's, and the programmer's perspective. *Sketch Understanding, Papers from the 2002 AAAI Spring Symposium* 73–77.

Landay, J. A., and Myers, B. A. 1995. Interactive sketching for the early stages of user interface design. In *Proceedings of CHI 95*, 43–50.

Lank, E.; Thorley, J. S.; and Chen, S. J.-S. 2000. An interactive system for recognizing hand drawn UML diagrams. In *Proceedings for CASCON 2000*.

Mahoney, J. V., and Fromherz, M. P. J. 2002. Handling ambiguity in constraint-based recognition of stick figure sketches. *SPIE Document Recognition and Retrieval IX Conf., San Jose, CA*.

Mankoff, J.; Hudson, S. E.; and Abowd, G. D. 2000. Providing integrated toolkit-level support for ambiguity in recognition-based interfaces. In *Proceedings of the CHI 2000 conference on Human factors in computing systems*, 368–375.

Newman, M. W.; Lin, J.; Hong, J. I.; and Landay, J. A. 2003. DENIM: An informal web site design tool inspired by observations of practice. *Human-Computer Interaction* 18(3):259–324.

Nielsen, J. Ten usability heuristics. In *useit.com: Papers and Essays*. http://www.useit.com/papers/heuristic/heuristic_list.html.

Saund, E.; Fleet, D.; Larner, D.; and Mahoney, J. 2003. Perceptually supported image editing of text and graphics. In *Proceedings of UIST '03*.