Sketching Interfaces: Toward More Human Interface Design

An interactive user-interface design tool supports electronic sketching, giving designers more freedom to change sketches and more flexibility in creating and evaluating a design prototype.

James A. Landay
University of California, Berkeley

Brad A. Myers
Carnegie Mellon University

As computers grow more powerful, less expensive, and more widely available, people are expecting them not only to perform obvious computational tasks, but also to assist in people-oriented tasks, such as writing, drawing, and designing. This shift is causing some user-interface (UI) researchers to rethink the traditional reliance on methods that are more machine-oriented and to look at ways to support properties like ambiguity, creativity, and informal communication. The idea is to bend computers to people's way of interacting, not the other way around.

This flexibility is particularly important in the early stages of UI design itself, when designers need the freedom to sketch rough design ideas quickly, the ability to test designs by interacting with them, and the flexibility to fill in the design details as they make choices.[1] Tools at this stage must support conceptual design, which is characterized by ambiguity and the need to create several design variations quickly, as the "Why Sketching Is Important" sidebar describes. Unfortunately, with current UI tools, designers tend to focus on issues such as colors, fonts, and alignment, which are more appropriate later in the design. Thus, most UI designers resort to sketching ideas on paper, but these are hard to edit and inconvenient for user evaluations.

Researchers at University of California, Berkeley and Carnegie Mellon University (CMU) have designed, implemented, and evaluated SILK (Sketching Interfaces Like Krazy), an informal sketching tool that combines many of the benefits of paper-based sketching with the merits of current electronic tools.

With SILK, designers can quickly sketch an interface using an electronic pad and stylus, and SILK recognizes widgets and other interface elements as the designer draws them. Unlike paper-based sketching, however, designers can exercise these elements in their sketchy state. For example, a sketched scrollbar is likely to contain an elevator or thumbnail, the small rectangle a user drags with a mouse. In a paper sketch, the elevator would just sit there, but in a SILK sketch, designers can drag it up and down, which lets them test the components' or widgets' behavior. SILK also supports the creation of storyboards—the arrangement of sketches to show how design elements behave, such as how a dialog box appears when the user activates a button. Storyboards are important because they give designers a way to show colleagues, customers, or end users early on how an interface will behave.

Designers can test the interface at any point, not just when they finish the design. When they are satisfied with their early prototypes, they can have SILK transform the sketch into an operational interface using real widgets, according to a particular look and feel.

SILK's design and implementation were based on a survey of practicing UI designers[2,3] and a review of the literature. Our research team also conducted a usability evaluation of the implemented system, involving both professional and student designers. The designers found SILK effective for both early creative design and for communicating the resulting design ideas to others. They were able to move quickly

0018-9162/01/$10.00 © 2001 IEEE

through several iterations of a design using gestures to edit and redraw portions of the sketch.

HOW SILK WORKS

As the designer sketches an interface using a set of components and gestures, SILK recognizes them and tells the designer what it believes she has drawn. Although SILK is designed for use with an integrated display tablet (stylus, tablet, and LCD), designers can also use a mouse or graphics tablet. The designer adds behavior through storyboarding—drawing arrows between related screens. After the designer tests the interface and iterates the design as needed, SILK transforms the rough design to a more finished looking implementation.

Recognition and annotation

SILK recognizes four primitive components—rectangle, squiggly line (to represent text), straight line, and ellipse. These are single-stroke shapes, which means that the designer must draw them without lifting up the pen or mouse button until the stroke is finished. These primitive components combine to form basic widgets. In

Why Sketching Is Important

Sketching and gesturing with a pen are two modes of informal, perceptual interaction that have been shown to be especially valuable for creative design tasks.[1] For designers, the ability to rapidly sketch objects with uncertain types, sizes, shapes, and positions is important to the creative process. This uncertainty, or ambiguity, encourages the designer to explore more ideas without being burdened by concern for inappropriate details such as colors, fonts, and precise alignment. Leaving a sketch uninterpreted, or at least in its rough state, is key to preserving this fluidity.[2]

In this early phase, ambiguity also improves communication, both with collaborators and the target audience of the designed artifact. For example, an audience examining a sketched interface design will be inclined to focus on the important issues at this early stage, such as the overall structure and flow of the interaction, while not being distracted by the details of the look.[3,4] When designers are ready to move past this stage and focus on the details, they can re-create the interface more formally.

Sketching on Paper

Several researchers have recognized the benefits that sketches provide. Frustrated by colleagues who mistook her early prototypes for more finished designs, Ying Ying Wong[4] of Apple sketched her designs on paper, scanned them into a computer, and used Macromedia's Director, a multimedia scripting tool, to add behaviors to the sketchy interfaces. Because the interfaces looked rough, her colleagues no longer mistook them for finished designs, but scanning the paper-based designs and then scripting their behavior was a lot of work.

Other researchers have found that sketches stimulate not only the clients or target audience, but also the designers to think more creatively. Vinod Goel[5] of UC Berkeley observed designers who were asked to solve design problems either by sketching on paper or by using a computer-based drawing program. When the designers generated a new idea in a free-hand sketch, they quickly followed it with several variations. But those who used a drawing program tended to focus more on refining the initial design, without generating design variations.

Electronic Sketching

An electronic sketching system like SILK has many advantages relative to paper-based sketching, as the main article describes. Key to the success of any such tool is its ability to recognize graphical elements common to a particular domain *as the designer draws them*. The designer can thus test the design at any point, which enables iterative design. The Electronic Cocktail Napkin[1] lets architects sketch designs on an electronic pad, recognizing the graphical elements common in architectural drawings. Another sketching system supports a free-form design environment.[6] Designers structure a sketch as a set of translucent, nonrectangular patches and attach dynamic interpretations so that the marks take on definitions or perform actions. For example, the designer can hand write a list of numbers and apply a calculator behavior to the list to add them up.

More recent work relies on standard mouse-based direct manipulation techniques for input while using fancy rendering schemes to make the output appear sketchy. The EtchaPad[7] drawing program, for example, uses a noise function to give drawings a less formal look.

References

1. M.D. Gross and E.Y. Do, "Ambiguous Intentions: A Paper-like Interface for Creative Design," *Proc. ACM Symp. User Interface Software and Technology*, ACM Press, New York, 1996, pp. 183-192.
2. M.A. Hearst et al., "Sketching Intelligent Systems," *IEEE Intelligent Systems*, vol. 13, no. 3, 1998, pp. 10-19.
3. M. Rettig, "Prototyping for Tiny Fingers," *Comm. ACM*, vol. 34, no. 4, 1994, pp. 21-27.
4. Y.Y. Wong, "Rough and Ready Prototypes: Lessons from Graphic Design," *Posters and Short Talks: Proc. Human Factors in Computing Systems*, ACM Press, New York, 1992, pp. 83-84.
5. V. Goel, *Sketches of Thought*, MIT Press, Cambridge, Mass., 1995.
6. A. Kramer, "Translucent Patches—Dissolving Windows," *Proc. ACM Symp. User Interface Software and Technology*, ACM Press, New York, 1994, pp. 121-130.
7. J. Meyer, "EtchaPad—Disposable Sketch Based Interfaces," *Proc. Conf. Companion on Human Factors in Computing Systems*, ACM Press, New York, 1996, pp. 195-198.

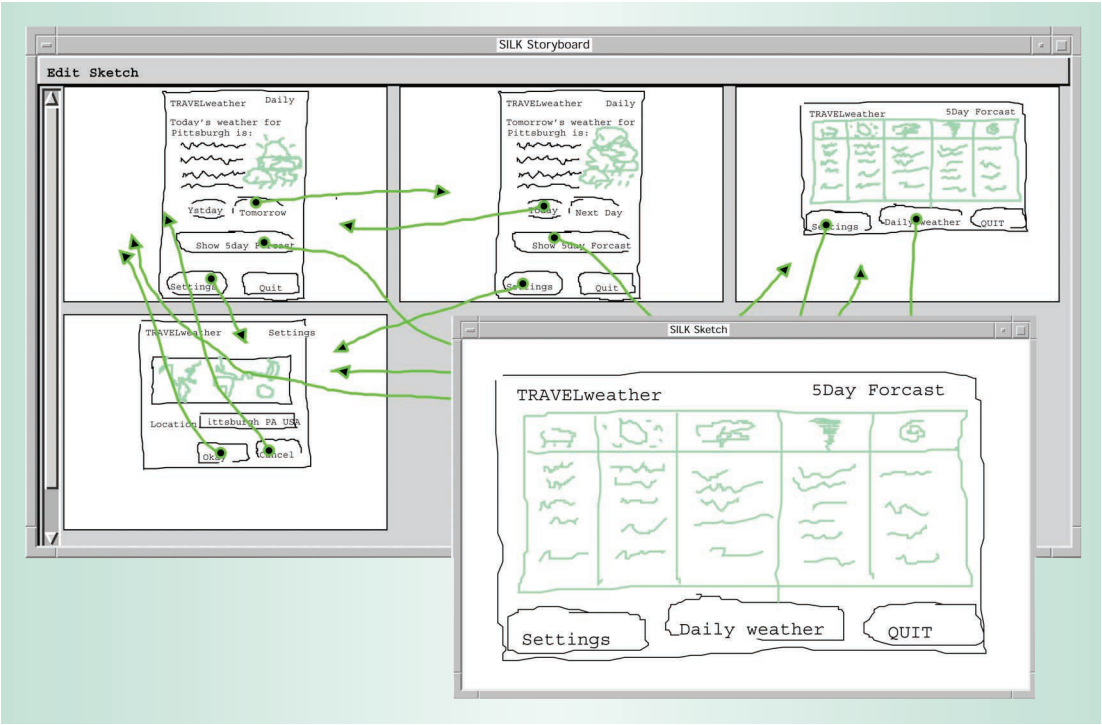

Figure 1. A SILK sketch (front) and storyboard (rear) for a weather application. An experienced user-interface designer created the sketch in about 30 minutes during a usability test. SILK recognizes common shapes such as ellipses and rectangles and provides feedback to designers about what it thinks they drew. The designer has sketched a screen that shows a five-day weather forecast (front). The designer has also created buttons below the forecast and drawn arrows from the first two (rear) so that when the user presses the buttons, he will go to the location settings dialog or see a weather report if he has set the right location. The ability to rapidly create storyboards like this, which lets both designers and prospective users see interface elements interacting, is one of SILK's main advantages over paper sketching.

Figure 1, for example, the designer sketched a rectangle and then drew a squiggly line inside, which SILK recognizes as the button widget. Later, the designer replaced the squiggly line with a text label, for example, "settings." SILK recognizes seven basic widgets as well as combinations of widgets, like panels, as Figure 2a shows.

As Figure 3 shows, SILK also recognizes editing gestures. When the user holds down the button on the side of the stylus, SILK interprets the strokes. Using gestures allows designers to specify, with a single mark, a set of objects, an operation, and other parameters. For example, deleting a section of the drawing is as simple as making an X-shaped stroke with the stylus.

Finally, SILK supports a mode for annotating sketches with drawn, written, or typed comments. The designer can display or hide the annotations, as desired. Practicing designers often view the annotations of design sketches as more valuable than the sketches themselves because the annotations serve as a diary of the design process.[4]

Behavior specification

Making it easier to specify the interface layout and structure solves much of the design difficulty, but there must also be some way to specify and evaluate the design's behavior. Because it automatically recognizes the behavior of the standard widgets, SILK lets designers or target users test a design in its sketchy, informal state. For example, as soon as SILK recognizes the buttons in Figure 1, the designer can switch to SILK's run mode and select one of the buttons with the stylus or mouse to show the user what the button does.

Knowing the behavior of one widget does not give a picture of the entire interface, however. For example, SILK knows how a button operates but cannot know what interface action should occur *after* the user presses that button. To illustrate this before-and-after behavior, designers use storyboards to simulate interface functions. Storyboards are a natural representation; they are easy to edit and designers need not worry about a particular form of implementation.

Transformation

When the designer is satisfied with the interface, SILK creates a new window that contains real widgets and graphical objects corresponding to those in

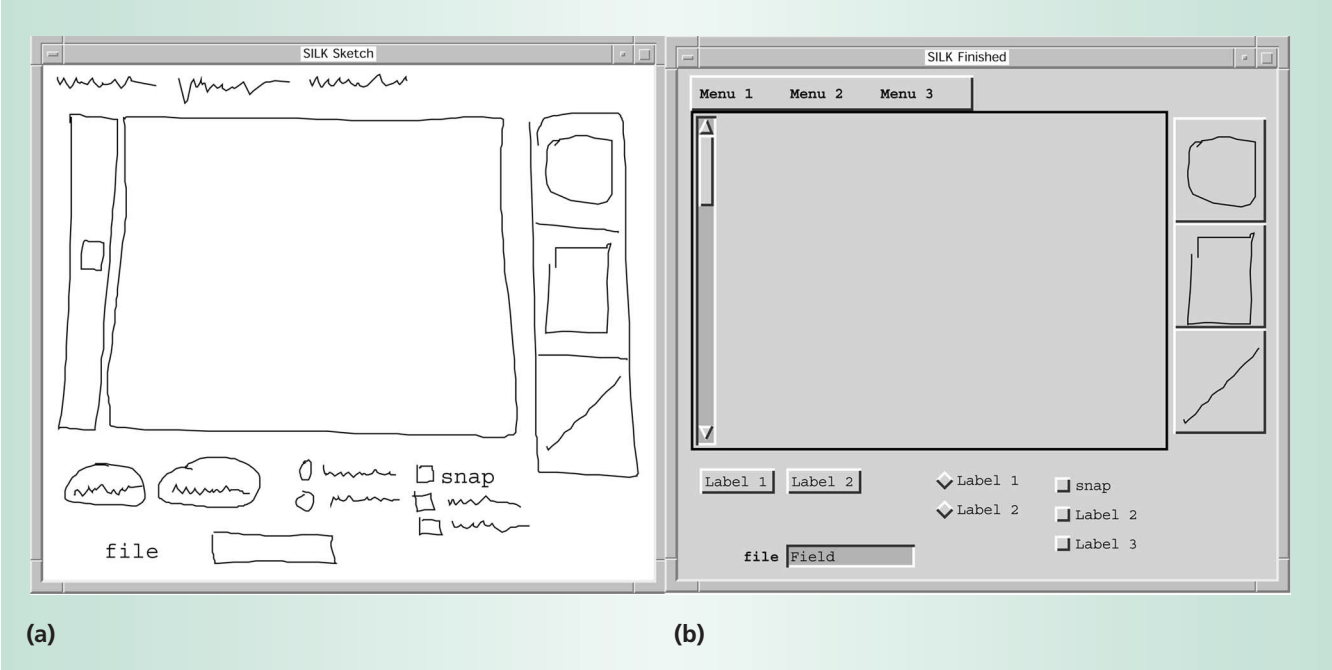

Figure 2. Interface widgets that SILK recognizes (a) during sketching and (b) in the transformed interface. From top to bottom, left to right: menu bar, scrolling window, palette, button, radio button, check box, and text field. SILK also recognizes vertical and horizontal sequences of some of these widgets as panels—for example, a check box panel, illustrated in the lower right. The sketched shapes in the palette on the far right were not transformed because they represent arbitrary bitmapped decorations. The designer can go back later and replace them with other images.

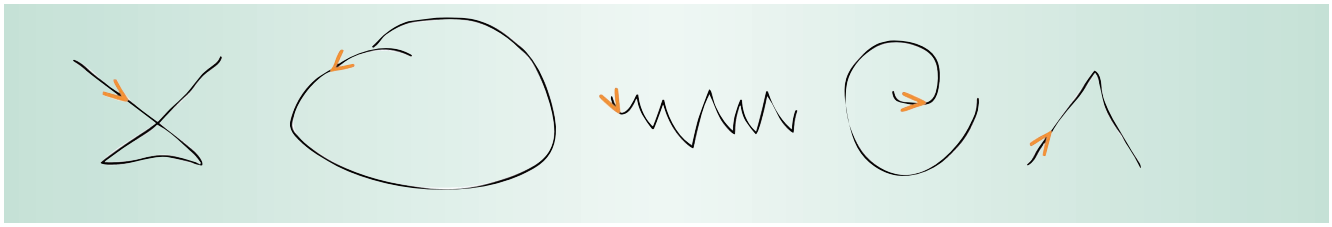

Figure 3. Editing gestures that SILK supports. From left, the first gesture deletes widgets or other objects in the sketch. The next three gestures help inform the recognition process: group objects, ungroup objects, and cycle to the next best inference. The last gesture lets the designer insert typed text or replace a text squiggle with typed text. The arrows show the direction to draw the gestures for best recognition.

the rough sketch. These objects take on the look of a specified standard graphical user interface. SILK currently outputs either Visual Basic 5 code or Common Lisp code using the Motif look-and-feel in the Garnet user-interface development environment.[5]

Figure 2b shows how SILK transforms the widgets in Figure 2a. The transformed interface is only partially finished because the designer still needs to finalize details such as colors, alignment, and any additional text labels. At this point, programmers can add callbacks and constraints that include the application-specific code to complete the application.

WIDGET RECOGNITION

To recognize a widget, SILK first attempts to identify primitive components using a gesture recognition algorithm. After it recognizes a primitive component, it looks for spatial relationships between the new component and other components in the sketch. Finally, it tries to identify the most likely widget that includes these components and rechecks the spatial relationships between the newly inferred widget and the rest of the components in the sketch.

Recognizing components

SILK recognizes gestures through Rubine's algorithm,[6] which uses statistical pattern-recognition techniques to train classifiers. The resulting classifier is based on features extracted from several examples. For SILK, we used 15 to 20 examples for each primitive component.

To classify an input gesture stroke, the algorithm computes its distinguishing features—angles and point-to-point distances, for example—and returns the best match with the learned gesture classes. This limits SILK to recognizing single-stroke primitive components, but we could eventually overcome this by combining independent strokes that occur within a specific timeframe or that are connected spatially.

SILK can also learn gestures that particular designers use to form the primitive components. Figure 4 shows SILK's control window. When SILK recognizes

SILK also provides feedback about its inference results so that designers can change the results when the system infers incorrectly.

a primitive component, it highlights the appropriate button in the window. If SILK doesn't recognize the component correctly, the designer merely clicks on the correct button, and SILK uses that type instead. From the points that compose the corrected stroke, SILK produces a new gesture classifier that more accurately recognizes the designer's way of drawing the primitive components. This retraining is transparent to the designer.

Detecting spatial relationships

As the designer sketches each component, SILK classifies it and passes it to an algorithm that looks for spatial relationships among both primitive and widget components. The algorithm asks

- *Does the new component contain, or is it contained by, another component?* This is the most important relationship for classifying widgets. Designers can express many common interface widgets via containment relationships among the widget's primitive components. For example, a scroll bar is a tall, skinny rectangle that contains a smaller rectangle.
- *Is the new component near (left, right, above, below) another component?* This relationship lets the algorithm recognize widgets such as check boxes, which usually consist of a box with text next to it.
- *Is the new component in a vertical or horizontal sequence of any combination of components that are the same type or are sequences of that type?* This relationship allows groupings of related components that make up a set of widgets, such as a panel of radio buttons.

Designers can specify hints to override the normal calculation of these spatial relationships. For example, many applications have a menu bar at the top left of the window but also locate some menus at the far right of the window. Unfortunately, sequences are defined as a series of the same object type such that each item is "near" the preceding item. This rules out a split menu bar, such as the Macintosh's help menu. Thus, the distance calculation ignores the distance between objects if the designer explicitly selects multiple objects at the time of inference. This is also useful when the designer sketches the text of a radio button or check box too far from the circle or rectangle.

Determining the intended widget

After identifying the basic relationships between the new component and the other components in the sketch, SILK passes the new component and the identified relationships to a rule system that uses basic knowledge of UI structure to infer which widget the designer intended. Each rule attempts to match the new component and relationships.

There is at least one rule for each widget that SILK recognizes, and each rule has two parts. The *test* part checks whether the rule applies. For example, the test for a vertical scroll bar makes sure that one component contains the other, that both components are rectangles, and that the container is skinny. The *then* part of the rule simply returns a list containing a confidence value for that match, the widget type, and a function that when evaluated can add the correct interactive behavior to the sketched components.

After the rule system tries each rule, it chooses the match with the highest confidence value and adds the proper interactive behavior to the sketched components. If no rules match, SILK assumes that there is not yet enough detail to recognize the widget.

SILK also provides feedback about its inference results so that designers can change the results when the system infers incorrectly. One form of feedback is that SILK draws the primitive components of widgets it recognizes in purple to indicate that the components are related. Designers can also look at the SILK controls window to see what widget SILK believes it has recognized. In Figure 4, for example, SILK has recognized a button. If this is not accurate, the designer merely selects another widget. If SILK made no inference on the widget in question, the designer can select the primitive components and click on the New Guess button in the controls window.

To use widgets or graphical objects that SILK does not recognize (other than those in Figure 2), the designer can switch from sketch mode to decorate mode and bypass the inferencing process.

STORYBOARD CREATION

Figure 1 is an example of storyboarding in SILK. Each sketch shows the interface in a particular state. Designers connect these sketches, or screens, by drawing arrows from any of one screen's graphical objects, widgets, or background to another screen. The arrow indicates that when the user clicks on the object from which the arrow originates, SILK should display the screen where the arrow points instead of the original screen. Thus, designers can simulate the changes on screen that will occur in the final interface. This visual representation, which designers can later view and edit, is much easier to use than the hidden-text representations of other systems, such as HyperCard.

Screens might differ only in the orientation of a primitive component, such as a rectangle. Designers can redraw the rectangle at a different angle in each succeeding screen to illustrate a behavior like rotation that the underlying tools, SILK and Garnet, do not even support. Designers can also hide underlying

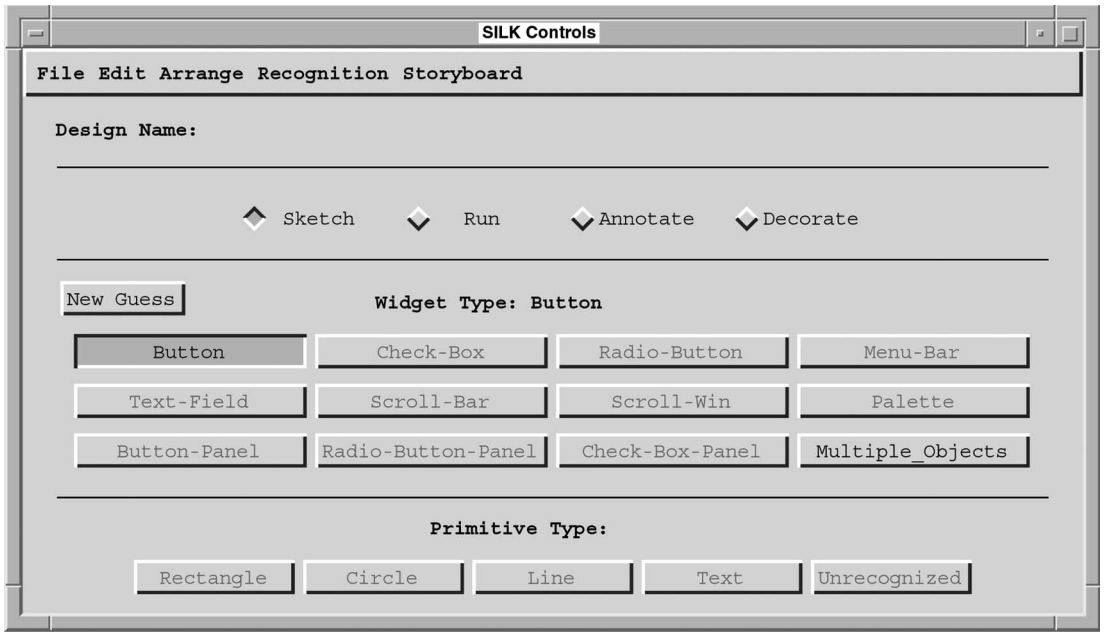

Figure 4. SILK's controls window. The designer can see what primitive component or widget SILK has recognized and correct the inference if needed. Here, the designer is in sketch mode, and SILK has recognized that the designer has sketched a button. The designer selects run mode to see how the interface widgets interact, annotate mode to add comments, and decorate mode to draw widgets or insert objects that are not in SILK's set of recognizable widgets (listed in Figure 2).

objects, which is useful in illustrating pull-down menus and dialog boxes.

Designers construct storyboards by sketching screens with a stylus or a mouse in the SILK sketch window (see Figure 1) and then copying the screens to the SILK storyboard window. After modifying the screens as desired, designers draw arrows to indicate screen sequencing and produce additional screens if needed.

When designers are ready to test the specified interaction, they switch to run mode. If desired, users can start interacting with the sketch. SILK displays feedback mechanisms while in run mode so that designers can debug their storyboards. For example, SILK highlights the active screen (the one in the sketch window) in the storyboard window. It also highlights the object that caused the last transition along with the arrow leading to the current screen.

MEASURING SILK'S USABILITY

To determine if UI designers would find an informal electronic sketching and storyboarding tool like SILK practical, we conducted a usability test to answer several questions:

- Can designers use SILK effectively to design user interfaces? Can they use it to produce more than trivial designs, and does it support their creativity by, for example, letting them work on more than one idea per session and focusing them on the creative process rather than on the detailed look and feel?
- Will SILK enable designers to communicate a design idea more easily with other design and development team members, as well as prospective users? Do these discussions concentrate on the structure and behavior of the interface, rather than the "look?" Can designers make changes immediately as a result of these discussions?
- How well will SILK perform? Do the recognition algorithms work? Can designers understand what SILK is doing? Is it easy to learn? How often does it perform the correct operation from the user's view?

The results of the study show that SILK is an effective tool for interface design, and the tool also provides an effective way of communicating design ideas to engineers. Designers found that learning SILK was not too difficult, although they noted the need for implementation and performance improvements in several areas.

Evaluation parameters

The evaluation involved six UI designers and six graduate students in computer science, robotics, or language technology, who acted as engineers. Four UI designers were "advanced"—had been practicing

An electronic sketching tool leads designers to focus on the overall interaction and structure rather than on the detailed look and feel.

interface design for more than a year. Two were "intermediate"—had taken at least one course that required significant work in creating actual interface designs. The designers were split evenly between male and female, with an average age of 30 years. All but one of the engineers were male, with an average age of 27.

We allowed four hours for the evaluation, which had five parts: an overview, a demonstration of SILK, a practice-run design task, a design task from which we took measurements, and a post-design discussion with the engineer to review and possibly change the design. After the discussion, we gave the designers a short questionnaire to evaluate whether they liked the underlying methodology or felt it would be practical and to capture any other comments.

Each designer used an HP Workstation (HP-735) with a three-button mouse. The designers received $25 each for their time, and the one with the best interface design, as judged by a CMU Human-Computer Interaction Institute faculty member, received an additional $100. The engineer participants received $8 each.

We asked designers to design an interface to a weather information system for travelers, a task based on a problem in *Usability Engineering*.[7] The finished interface was to provide information about the weather for the current day and predict the weather for the next two days. The designers' stated goal was to explore the possible design space and eventually present several good alternatives to the rest of their design team or to a client.

We took several measurements for each design session by observing the participant's actions, automatically logging events in SILK, videotaping, and having the participants fill out a post-evaluation questionnaire. We also used a set of measures tailored to our three objectives.

Design effectiveness

To measure SILK's design effectiveness, we looked at the time to complete the task, the formality and complexity of the designs produced, how often designers used the run mode, and SILK's capabilities relative to other tools.

Each designer spent about 1.5 hours on the task and came up with two designs on average. The resulting sketches were nontrivial, and the designers did not fixate on a single design idea. Overall, they felt that SILK's "ability to prototype screen-based interactions is GREAT," but that the tool needed better support for visual effects (such as color and multiple type sizes) and more support for nonstandard interactions.

The designs varied both between designers and for an individual designer. On average, the designs used six screens, with two transitions per screen. Four designers added interaction for objects that SILK did not recognize—another sign that the designs were not trivial. This, together with the variation in designs, led us to conclude that SILK does indeed support creativity in the early design stages.

Focus on creativity. Participants tended to leave parts of the interface in an especially rough and ambiguous state. For example, they did not spend much effort trying to align objects exactly but put things roughly where they wanted them and moved on with the design. They also left text rough until they had more details or until the end of the design session. Half of the designers liked the roughness or paper-like sketching that SILK provides. They appreciated the "ability to be fast and sloppy" and the "way it remains sketchy."

Most of the designers used cut, copy, and paste to reuse widgets or portions of screens and to revise individual designs. These editing features are another advantage of electronic over paper-based sketches.

Overall, the designers were fairly positive about SILK as a tool for early creative design, describing it as "an excellent first draft tool" that would be "useful for quickly structuring information." One designer said that SILK "is great for giving the idea of a progression through a program without getting into the details of the visual design." A few designers still preferred pencil and paper but found that SILK was similar in many ways.

These observations and comments confirm the hypothesis that an electronic sketching tool leads designers to focus on the overall interaction and structure rather than on the detailed look and feel.

Storyboarding. The designers typically used storyboards to illustrate a few important sequences. One designer wrote that SILK was "as quick as paper sketching and provides a basis for interaction." After running their designs, designers would often notice that they had left out necessary transitions, objects, or screens. They would then make the necessary changes and test it again. One designer said she "liked the ease with which you could test the interaction—it's a very tight loop."

Two designers confirmed that SILK let them illustrate behavior that the underlying tools do not directly support. For example, they repeatedly implemented their own state feedback on unrecognized radio buttons. The other designers also liked the ability to "see and edit the storyboard." Another wrote, "better than Director—the linking with drawings rather than Lingo [Director's scripting language] is excellent." Designers continually referred positively to the interaction and navigation possible via storyboards. All but one designer saw storyboarding as SILK's advantage over Director and HyperCard.

Perceived problems. All but one designer used SILK's built-in widgets in their designs, but recognition accu-

racy was low enough to be a hindrance. One designer felt that the "widget detection and sketching allows for fast low-level interaction," but we observed others become frustrated when faced with repeated recognition errors.

Communication effectiveness

To measure communication effectiveness, we looked at whether the post-design discussion focused on structure and behavior. We also evaluated the use of storyboarding, editing, and run mode during the discussion.

Three designers used the storyboard during the discussion to illustrate the design's overall structure. All but one designer used run mode to exercise the interface during the design review.

Four engineers asked critical questions. One asked; "What is the most direct way to move to a city that is not on the map?" This illustrates that the engineers understood the interface designs and were looking for places where the ideas may have needed more thought. We observed that these conversations concentrated on the interface's structure and functionality, not on its visual details.

Finally, half the designers made real-time changes to their designs during the discussions. One designer used the engineer's feedback—too many mouse clicks to get the desired information—to rapidly modify his first design. The modifications, which involved combining screens, took under five minutes, so the designer could make them while the engineer was there.

These examples show that SILK successfully achieved the objective of letting designers effectively communicate a design idea to other members of a design team.

Performance

To measure SILK's performance, we looked at how well the gesture and widget recognition algorithm performed and reviewed the designers' comments.

All the performance results have some ambiguity because we could not always know what the designers were trying to draw. For example, we know only the percentage of primitive components that SILK recognized minus the percentage that the user explicitly corrected. We don't know of others that were wrong but that the user never bothered to correct. However, we still believe the overall results represent a good estimate of SILK's practical application performance.

Overall, the designers rated SILK a 6.2 on a scale of 0 (worst) to 10. The main criticisms were of the implementation specifics and the "UI of the tool itself."

Gesture recognition. SILK recognized editing gestures about 89 percent of the time. Five designers stated that the editing gesture recognition worked "well" or "OK." This rate might improve with the use of a stylus rather than a mouse. The recognition rate for primitive component gestures was 93 percent. We considered an error to be when the designer had to correct SILK's component inference.

Widget recognition. On average, SILK recognized the correct widget only 69 percent of the time. Designers repeatedly failed to notice that SILK had earlier misrecognized one of the primitive components they now wanted grouped into a widget. For example, designers often drew buttons that were wider than SILK's rules permitted. SILK's failure to provide sufficient feedback about its recognition was the source of most of the confusion over the widget recognition algorithm. The rate might improve if SILK could learn new rules for inferring widgets. On the other hand, *any* widget recognition algorithm might be too error-prone and thus hurt the design process more than it helps. If this is true, a more explicit widget typing technique may be called for.

Other implementation issues. The designers encountered several problems with SILK's interface. They misunderstood the sketch-storyboard relationship, how to enter and manipulate typed text, and how to select, group, and move objects. The problems with selection had to do with SILK's nonstandard mapping of mouse buttons to editing and drawing functions.

SILK successfully achieved the objective of letting designers effectively communicate a design idea to other members of a design team.

The usability test showed that electronic sketching effectively supports the early stages of UI design. On average, designers produced two designs in about an hour and a half. Despite the problems the usability test revealed, particularly widget recognition, SILK is still a promising tool for early UI design. As one designer put it, "SILK works like pencil and paper; is simple, and shows the logic of navigation." The designers finished the SILK tutorial in less than an hour and were able to use all of SILK's major features.

Addressing the widget recognition problems will require more research. SILK could learn a widget by learning the spatial relationships between its primitive components, as it already computes these relationships to recognize widgets, and adding these to its rule systems. Bayesian belief networks[8] could help SILK's rule system learn better confidence values. By noticing how a designer corrects its widget inferences, SILK could adjust its probabilities and make better inferences.

But even with its current recognition accuracy, SILK is an effective informal design tool for a specific creative task. Providing this kind of tool was the main goal in creating SILK, and UC Berkeley researchers are now working on several similar tools for other

creative tasks. Denim,[9] for example, lets Web site designers quickly prototype some of a site's more interactive portions. Suede is a speech interface prototyping tool that lets designers rapidly create prompt/response speech interfaces.[10] It offers an electronically supported Wizard of Oz technique that captures test data, allowing designers to analyze the interface after testing. Even nonexpert designers of speech UIs can quickly create, test, and analyze prototypes.

Finally, Berkeley researchers have incorporated some of SILK's ideas into a communication tool. NotePals[11] is a lightweight meeting support system that automatically combines individuals' meeting notes into a shared meeting record. Users take notes in digital ink, which frees group members from having to learn a shorthand method or to continually correct a handwriting recognition engine.

All these projects (available for download at http://guir.berkeley.edu/projects) are a step toward letting designers and others concentrate on the creative part of their work—without forcing them to use solutions that are suitable only when precision is necessary. ✷

References

1. A. Wagner, "Prototyping: A Day in the Life of an Interface Designer," *The Art of Human-Computer Interface Design*, B. Laurel, ed., Addison-Wesley, Reading, Mass., 1990, pp. 79-84.
2. J.A. Landay, "Interactive Sketching for the Early Stages of User Interface Design," PhD dissertation, tech. report CMU-CS-96-201, CS Dept., Carnegie Mellon Univ., Pittsburgh, Pa., 1996; http://www.cs.berkeley.edu/~landay/research/ publications/Thesis.pdf.
3. J.A. Landay and B.A. Myers, "Interactive Sketching for the Early Stages of User Interface Design," *Proc. Human Factors in Computing Systems*, ACM Press, New York, 1995, pp. 43-50.
4. D. Boyarski and R. Buchanan, "Computers and Communication Design: Exploring the Rhetoric of HCI," *Interactions*, vol. 1, no. 1, 1994, pp. 24-35.
5. B.A. Myers et al., "Garnet: Comprehensive Support for Graphical, Highly-Interactive User Interfaces," *Computer*, Nov. 1990, pp. 71-85.
6. D. Rubine, "Specifying Gestures by Example," *Computer Graphics*, vol. 25, no. 3, 1991, pp. 329-337.
7. J. Nielson, *Usability Engineering*, Academic Press, Boston, 1993, pp. 272-275.
8. J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann, San Francisco, Calif., 1988.
9. J. Lin et al., "DENIM: Finding a Tighter Fit between Tools and Practice for Web Site Design," *Proc. CHI Conf. Human Factors in Computing Systems* (CHI00), vol. 2, no. 1, 2000, pp. 510-517.
10. S.R. Klemmer et al., "SUEDE: A Wizard of Oz Prototyping Tool for Speech User Interfaces," *CHI Letters: Proc. ACM Symp. User Interface Software and Technology*, vol. 2, no. 2, 2000, pp. 1-10.
11. J.A. Landay and R.C. Davis, "Making Sharing Pervasive: Ubiquitous Computing for Shared Note Taking," *IBM Systems J.*, vol. 38, no. 4, 1999, pp. 531-550.

James A. Landay *is assistant professor of computer science at the University of California, Berkeley. He is also the chief technical officer and cofounder of NetRaker. His research interests include UI design tools, gesture recognition, pen-based UIs, mobile computing, and Web-site evaluation tools. Landay received a PhD in computer science from Carnegie Mellon University. Contact him at landay@cs.berkeley.edu.*

Brad A. Myers *is a senior research scientist in the Human-Computer Interaction Institute in the School of Computer Science at Carnegie Mellon University, where he is researching UIs for hand-held devices, making programming more accessible, UI software, and programming by example. Contact him at bam@cs.cmu.edu.*