

# User Interfaces for On-line Diagram Recognition

Dorothea Blostein, Ed Lank, Arlis Rose, Richard Zanibbi  
Dept. Computing and Information Science  
Queen's University, Kingston Ontario, Canada, K7L 3N6  
blostein@cs.queensu.ca lank@cs.sfsu.edu zanibbi@cs.queensu.ca

The user interface is critical to the success of a diagram recognition system. It is difficult to define precise goals for a user interface, and even more difficult to quantify performance of a user interface. In this paper, we discuss some of the many research questions related to user interfaces in diagram recognition systems. We relate experiences we have gathered during the construction of two on-line diagram recognition systems, one for UML (Unified Modeling Language) notation and the other for mathematical notation. The goal of this paper is to encourage discussion. The graphics recognition community needs strategies and criteria for designing, implementing, and evaluating user interfaces.

**Keywords:** diagram recognition, graphics recognition, user interfaces

## 1. Introduction

User interfaces are challenging to design and construct. The success of the user interface has an enormous influence on the success of recognition software. In this paper we discuss issues that arise in the construction of user interfaces. We draw on our experience in implementing two on-line diagram recognition systems, one for UML<sup>1</sup> notation [17] and the other for mathematics notation [29] [30]. Both of these build on the character recognizer and user interface of Smithies, Novins and Arvo [26].

### 1.1 Existing Research into User Interfaces

Human Computer Interaction (HCI) has been studied extensively [8]. This literature provides insights relevant to the construction of user interfaces for diagram recognition systems. Selected papers are discussed in the following sections.

Software tools that support user-interface construction are reviewed by Myers [21]. Application of these tools to diagram recognition systems should be investigated. However, generic tools may be difficult to apply: diagram recognition requires close interaction between the user interface and the recognition algorithms. The user interface should provide the user with the highest-quality recognition results for a given amount of user input time. The user's input time can be spent before recognition (e.g., training a recognizer), during recognition (e.g., inspecting and correcting intermediate results), or after recognition (proofreading and correcting recognition errors).

HCI researchers have found that system design requires a deep understanding of the user's tasks [11] and the user's physical context [12]. Scenarios of use [7] can be developed to show the broader context of diagram use. For example, a diagram may be drawn as part of a brainstorming session. This requires a fluid drawing environment, in which users are not inhibited by requirements of syntactic correctness or complex editing techniques [27]. In contrast, the creation of a precise diagram (e.g., an

---

<sup>1</sup>UML, the Unified Modeling Language, is a diagram notation for describing the structure of software systems. Early in the software design cycle, software engineers informally sketch UML diagrams on paper or whiteboards. UML recognition software can make this diagram information available to Computer Assisted Software Engineering (CASE) tools.

engineering schematic, or a blueprint for a building) may be best served by tools that do not use on-line document recognition, but instead provide structured document entry with checking and analysis facilities.

If an on-line diagram recognition tool is used to support brainstorming, the tool must support collaborative work. This means that the system must be able to sort out whether strokes drawn by a pair of users are contributing to a single drawing element or to separate elements. While a single person can work with a small drawing surface, such as a piece of paper or a small computer monitor, group work requires a large physical drawing surface, such as a whiteboard or flip chart. Various activities accompany the drawing of a diagram, including gesturing, pointing, annotating and list-making. It is challenging for the user interface of an on-line diagram recognition tool to permit all of these activities; in addition the user must be able to easily specify what parts of the drawing are candidates for recognition.

### **1.2 On-line versus off-line recognition**

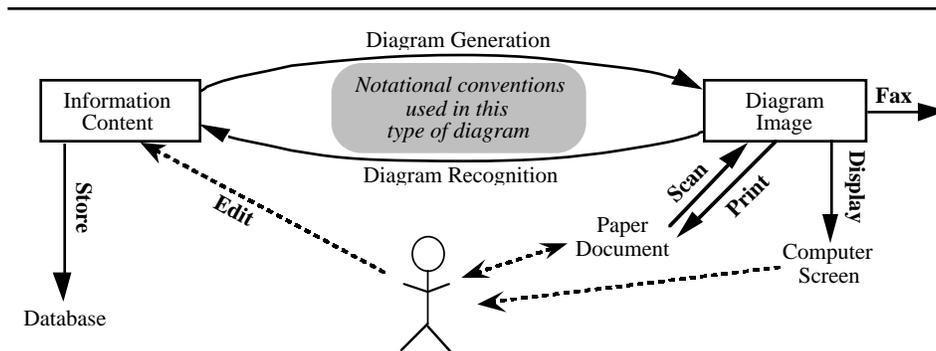
Our focus is on user interfaces for on-line diagram recognition systems. Here we briefly contrast on-line and off-line systems. The user interface is pervasive in an on-line system, responding to the strokes the user draws. In contrast, an *ideal* off-line system has a minimal user interface: a user presents a stack of paper to the scanner, and the recognition system processes this fully automatically, with no recognition errors. In reality the recognition results are not perfect, so a user interface is provided to support proofreading and error correction.

On-line systems process modest amounts of data, whereas off-line systems can be applied to large-scale data acquisition tasks. This difference in data volume has a significant impact on user interface requirements. In the on-line case, the user draws diagrams while connected to the computer, and is available for proofreading and correcting the recognition results. In the off-line case, the large volume of data may mean that the user cannot invest the time to proofread and correct the recognition results. This problem can be addressed both by developing highly accurate recognition algorithms, and by finding ways for subsequent software to use noisy recognition results. For example, noisy OCR results are used in the text editing system of [2]: the scanned document image is presented to the user, with the noisy OCR result hidden from sight but used for content searches.

### **1.3 The Interaction Between Editing, Generation and Recognition**

Diagram recognition has traditionally been well-separated from diagram generation and editing. This arises naturally in off-line diagram recognition, where the goal is to get a one-time transfer of information from paper documents to an electronic form. As discussed below, on-line diagram recognition can lead to a closer relationship between recognition, editing and generation.

Figure 1 illustrates the image and information aspects of a diagram [4]. Diagram *generation* is a translation from information to image, whereas diagram *recognition* is a translation from image to information. In traditional off-line diagram recognition systems, a recognizer is applied once, to transform the image into information. This is followed by an *edit/generate cycle*, in which the user iteratively edits the information and views a re-generated diagram. The user edits both to correct recognition errors, and to change the diagram. For example, recognized music notation may be transposed; this creates new sheet music to fit the vocal range of a singer.



**Figure 1** A diagram must be treated as both image and information. The image is displayed to the user, and supports image-level operations such as faxing. The information supports intelligent editing operations, content-based searches, and many other operations which require an understanding of the diagram.

Diagram editors provide either a batch or WYSIWYG (What You See Is What You Get) user interface. In a batch system, such as LaTeX, the user directly edits a textual representation of the information, and requests image generation when desired. In a WYSIWYG system, the user views the diagram image, issues editing commands to update the information, and views an updated image created by the diagram generator [3]. The WYSIWYG interface does not require the user to learn a textual encoding of the diagram information.

On-line diagram recognition offers the possibility of more closely integrating the recognizer into the edit/generate cycle. A user begins by drawing a portion of a diagram, which is recognized and perhaps rectified. The user continues to edit: correcting recognition errors, moving and resizing diagram elements, and drawing new parts of the diagram. This scenario puts complex requirements on the user interface for the diagram recognition system. It must be possible to apply recognition algorithms to the new parts of the diagram, while preserving the recognition results (and user corrections) made to the old parts of the diagram. Ideally, the user should have one integrated piece of software which provides diagram recognition, diagram editing and diagram generation capabilities.

#### 1.4 The User Interface Provided by Paper

Paper is popular and widely used. Before designing a user interface for document recognition, it is worth reviewing the properties of paper that make it so attractive: ergonomics, contrast, resolution, weight, viewing angle, durability, cost, life expectancy, and editorial quality [15]. Paper also has limitations: erasing is difficult, it is not possible to “make extra room” to expand parts of a diagram, it is hard to find information in a large stack of paper, and so on. A goal for future computer interfaces is to retain the advantages of paper while also providing the editing and search capabilities lacking in paper.

Gross and Do have observed that designers prefer to use paper and pencil [13]. In particular, designers reject the use of computers in the early, conceptual, creative phases of designing. Paper and pencil permits ambiguity, imprecision, and incremental formalization of ideas. With paper and pencil, you draw what you want, where you want it, and how you want it to look. In contrast, computer based tools force designers

into premature commitment, demand inappropriate precision, and are often tedious to use when compared with pencil and paper.

The rest of this document discusses user interface issues related to quantifying the performance of a user interface (Section 2), supporting diagram input (Section 3), executing the recognizer (Section 4), displaying recognition results (Section 5), and supporting user correction of recognition errors (Section 6). Important research problems exist in all of these areas.

## **2. Quantifying the Performance of a User Interface**

The performance of a user interfaces is difficult to measure. One challenge is to separate the performance of the user interface from the performance of the recognition algorithms. (Evaluation of graphics recognition algorithms is discussed in [25]). The user interface plays a big role in determining how much time the user spends finding and correcting recognition errors. It is particularly difficult to develop a quantitative performance measure for incremental recognition, where the user can intervene at different steps of the design [16].

### **2.1 Comparing Automated versus Unautomated Diagram Entry**

One approach to performance measurement is to compare automated diagram entry (using diagram recognition software) to unautomated diagram entry (where the user redraws the diagram using a structure-based editor, or the user directly types the information, as in LaTeX). Analogous measurements have been made for OCR, comparing the time needed to type and proofread a page of text to the time needed to proofread and correct OCR results produced for this page of text [9]. The conclusion was that OCR must be at least 98% correct to match the data entry time and accuracy (number of residual errors after proofreading) achieved by human typists. Unfortunately, there are no comparable results for diagram recognition. We would like to compare diagram entry (user time taken, residual errors) using two methods:

1. *Automated diagram entry*: On-line or off-line diagrams are processed by recognition software, producing information which is proofread and corrected by the user.
2. *Unautomated diagram entry*: A user enters diagram information via editing commands (using a batch or WYSIWYG editor, as discussed in Section 1.3).

The user time taken for method 1. depends both on the recognition algorithm (how many recognition errors there are for the user to correct) and on the user interface (how quickly the user can find and correct these errors). The performance baseline provided by method 2. is well-defined for text entry, but may change for diagram entry: the speed of skilled human typists is not likely to increase much in the next decade, whereas the speed of skilled human entry of diagrams (using method 2.) could increase due to advances in diagram editors. Since many users are interested in finding the fastest and most accurate method of entering diagram data, there is a real need for quantified comparisons of automated and unautomated diagram entry.

### **2.2 Other Performance Measures for User Interfaces**

The previous section discusses a performance measure based on user entry time. That is one useful measure of performance, but additional measures are needed to capture other performance aspects. Can a user draw a diagram in a comfortable, freehand, unconstrained way? This is particularly important when the user does not start with a paper diagram, but instead is creating a diagram interactively. Also, how

frustrating does a user find the recognition system? Currently, diagram editing and generation software tends to be less frustrating (and more predictable) than diagram recognition software. Quantifiable performance metrics are needed for these aspects of a user interface.

Nielsen has defined usability as consisting of five attributes: learnability, efficiency, memorability, errors and satisfaction [23]. Measuring these attributes is difficult. Nielsen and Levy discuss two categories of measurable usability parameters [24]: subjective user preference measures, which assess how much the users like the system, and objective performance measures, which measure the speed and accuracy with which users perform tasks on the system. Although users sometimes prefer the system on which they perform more poorly, in most cases preference and performance are positively associated [24]. Desirable usability principles listed in [14] include control, predictability, transparency, trust and privacy.

The *usefulness* of a design refers to the possibilities for action in the design, whereas *usability* refers to the user's ability to notice these possibilities for action [20]. Usability depends on the user's experience level. For example, a standard GUI provides extensive information about the set of actions that are available to the user. This is of great help to novice users. Expert users tend to prefer command-line interfaces: they have memorized the commands, and find it faster to enter a short keyboard command than to move the hand to the mouse, position the pointer, and click [20].

### 3. Supporting Diagram Input

Pens are commonly used as input devices for drawing diagrams. A pen is natural and easy to use on small surfaces (data tablets) and large surfaces (electronic whiteboards). A two-ended pen supports fast mode switching (Section 6). Perhaps the use of a programmable data glove could be investigated: all fingers can be involved, providing new ways of drawing, selecting objects, and correcting recognition results.

Diagram entry can occur through unconstrained drawing, or via gestures. Gestures are stylized pen strokes which invoke a command. In gesture-based entry, there is no need for diagram recognition – only gesture recognition is required. As an example, the Knight Project supports gesture-based creation of UML diagrams [10]. Unconstrained diagram entry is more natural (more like drawing on paper) than gesture-based diagram entry. However, recognition performance must be high. Otherwise users are likely to prefer the more constrained but more reliable entry method based on gestures.

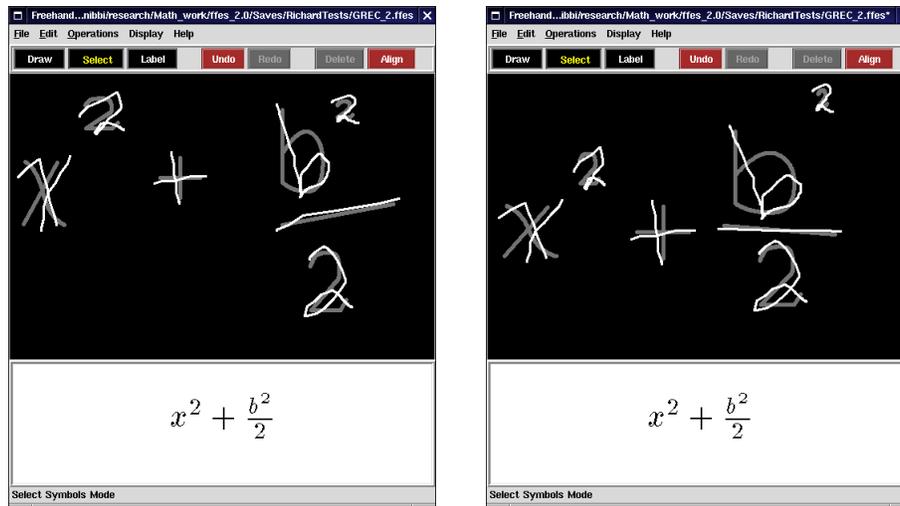
In [16], gestures used for drawing correction include erase (back-and-forth scribble), move (arrow) and select (enclosing circle). In trials on nine novice users, only 74% of 269 erasing gestures were effective; surprisingly, this did not discourage the users, who always chose to gesture when they had a choice between gesture and menu for erasing. There is need for standardization of gestures for common operations; existing software uses various gestures for the same operation [18].

### 4. Executing the Recognizer

Recognition can be *eager* (occurring while the user draws), or *lazy* (occurring only when explicitly requested by the user, or when recognition results are required by another operation). Lazy recognition prevents the recognition process from intruding on the user's creative phase [22]. In contrast, when eager recognition is combined with immediate changes to the display, the user may be distracted. For example, eager

rectification of a user's strokes can be disorienting, because items that the user just drew suddenly change appearance.

Eager recognition is used in our mathematics recognition system (Figure 2). Every time the user draws a new math symbol in the drawing panel, the recognizer produces an updated LaTeX interpretation. This relies on the ability of the recognition algorithms to process incomplete expressions [29].



**Figure 2** These screen shots from our mathematics recognition system [29] show three different methods of providing feedback about recognition results. (1) The lower panel displays the LaTeX output produced by the recognizer. Eager recognition is used, updating this panel every time the user enters or moves a character. (2) The upper panel shows character-recognition feedback provided by superimposed drawing annotations. (3) The left screen shot shows the original input, as written by the user. The right screen shot shows the result after a style-preserving morph (Section 5.3). The morph gradually moves and resizes symbols to display the baselines perceived by the recognizer.

Lazy recognition is used in the Burlap system [19]. The meaning of a sketch is not recognized until the user actually tries to interact with the sketch. In other systems, recognition occurs after a sufficiently long drawing pause. For example, in the Electronic Cocktail Napkin, configuration recognizers are run when the user pauses more than five seconds. These recognizers search the drawing for instances of their patterns [13].

Many different types of diagrams are used in society, so it is useful if a diagram recognition system is able to accept multiple diagram types. The user can be asked to identify the diagram type, or the system can infer this information. Our UML recognition system currently handles Class, Use Case, and Sequence diagrams, with the user indicating ahead of time which type of diagram is being drawn [17]. The Electronic Cocktail Napkin provides definition and inference of diagram types, called *contexts* [13]. The user gives examples to define graphical rewrite rules (patterns and replacements). These are used to recognize configurations in a drawing. In one example, the pattern is a large rectangle surrounded by smaller squares, and the replacement is the configuration *table surrounded by chairs*. This configuration is

defined in the Room context, and does not apply in another context such as Circuits. The system identifies the context when the user draws a symbol or configuration unique to one context. For example, a resistor symbol is found only in the Circuits context.

## 5. Displaying Recognition Results

Diagram recognition systems need to display a variety of information. Our UML recognition system [17] directly or indirectly displays: the drawing made by the user, the segmentation of strokes into glyphs, the classification of glyphs into “UML glyph” versus “alpha-numeric character”, the results of character recognition and UML glyph recognition, and a rectified version of the drawing. Recognized glyph relationships are not explicitly displayed, but are used in glyph moving (e.g., if the user moves a box, attached arrows move as well). As another example, our mathematics recognition system [26] [29] [30] displays: the drawing made by the user, the segmentation of strokes into glyphs, character recognition results, a partially rectified version of the drawing, the recognition result as raw LaTeX and as typeset math notation produced from the LaTeX. The partial rectification uses a style-preserving morph [30]: characters and symbols are moved and resized, but retain the shape of their handwritten strokes. This gives the user feedback on the recognition of superscript, inline and subscript relations.

As discussed below, mechanisms for displaying recognition results include multi-view displays, superimposed drawing annotations, rectification, and morphing. Extra thought must go into the display of alternative or ambiguous recognition results.

### 5.1 Multi-view Displays

Multi-view displays [5] [6] show several views of related data, such as the drawn diagram and the recognition result. A two-view display can also be used to reduce input ambiguity, where user-drawn strokes can be categorized either as a command or as graphic input [16]. For example, if the user draws a circle around an object, the intent may be to select the object (a command) or to add the circle to the drawing (graphic input). A two-view display avoids this ambiguity by providing a drawing window for graphic input, and a correction window (which shows recognition results as a rectified drawing) for commands.

### 5.2 Superimposed Drawing Annotations

Superimposed drawing annotations (displayed in a contrasting colour) can be used in various ways. The results of glyph recognition can be presented via superimposed bounding boxes and labels. However, this gives the display a cluttered appearance. We have developed a neater, more readable way of displaying character recognition results, as shown in Figure 2. Lightly-coloured typeset symbols are placed behind the user’s handwritten symbols. The typeset symbols are sized and scaled to fit in the bounding box of their associated handwritten symbol. Initial results are encouraging, with both handwritten and typeset symbols clearly visible.

The Electronic Cocktail Napkin uses superimposed drawing annotations to provide recognition and constraint feedback [13]. For example, a constraint [28] can cause one node to move whenever the user moves another node. The Electronic Cocktail Napkin displays constraints as superimposed drawing annotations, analogous to dimensioning annotations in mechanical engineering drawings. In conclusion after user trials, Gross and Do recommend that The Electronic Cocktail Napkin be run with low-level echoing and rectification turned off, to avoid distracting the user.

### 5.3 Rectification and Morphing

Rectification replaces drawing elements by stylized versions of their shapes. This provides useful feedback, but a user can be disoriented by the sudden change in drawing appearance. *Morphing* reduces this problem by displaying a series of images that show a smooth transition from the input to the rectified feedback [1]. Our mathematics recognition system uses a style-preserving morph in which the shape of the handwritten strokes is preserved; the symbols are gradually moved and resized to display the baselines that make up the structural interpretation of the mathematical expression (Figure 2). An experiment compared user performance with morphing feedback to user performance with typeset feedback shown in a separate window [30]. Expression-entry (and correction) time was equally good under both conditions. This is despite the fact that the style-preserving morph does not provide explicit symbol recognition feedback, and that the morphing itself introduces a delay of about one second. The success of the style-preserving morph may be due to reduced disruption of the participants' mental map; the typeset feedback is more disruptive because participants have to shift focus between the drawing window and the window that gives feedback via a typeset expression.

### 5.4 Treatment of Ambiguity

Mankoff et al. provide an interesting discussion of user-interface issues arising from recognition ambiguity [19]. Their goal is to create a toolkit supporting creation of effective user interfaces for software which involves recognition. Recognition errors easily confuse the user, and result in brittle interaction dialogues. Existing user interface toolkits have no way to model ambiguity. Mankoff et al. address symbol recognition ambiguity, segmentation ambiguity, and target ambiguity. (Target ambiguity occurs when it is unclear which drawing elements are the target of a user's selection operation.) Two design heuristics for increasing the usability of mediators that handle recognition ambiguity are described. The first heuristic is to provide sensible defaults, for example highlighting the top choice in an N-best list that is displayed to the user. The second heuristic is to be lazy. Put off making choices; these can result in errors. Later input may provide information that helps with disambiguation, and some input (such as comments and annotations) may never need to be disambiguated.

## 6. Supporting User Correction of Recognition Errors

Good user interface support is needed to allow the user to quickly locate and correct recognition errors. In some cases, domain-specific methods can be applied to help locate errors. For example, a recognition system for music notation can use audio feedback to allow the user to listen for recognition errors. A mathematics recognition system can use numerical evaluation to check the correctness of recognized equalities.

Modes are commonly used to support error correction. Our UML and mathematics recognition systems use diagram-entry and diagram-correction modes. The user clicks a button to change modes. More natural mode selection might be achieved by a two-ended pen. People are accustomed to using the two ends of a pencil for writing and erasing. The two pen ends can be interpreted in a variety of ways, perhaps using one end for drawing the diagram and the other end for commands and corrections. The user need to know which mode is currently selected. Mouse-based systems can use cursor shape to indicate mode. This does not carry over to our systems, because the electronic

whiteboard does not provide cursor tracking when the pen is away from the drawing surface. We continue to investigate ways to handle modes in our recognition systems.

Overtracing can be used to correct glyph recognition errors [13]. If the user draws a blob that could be either 'circle' or 'box', both interpretations are retained. The user can later identify the element by drawing over it, this time more clearly as a box.

N-best lists have been widely used to allow a user to select from a set of recognition alternatives. Uses of N-best lists include the following [19]: a voice recognition system uses an N-best list to display candidates for a misrecognized word; a drawing system displays an N-best list for a misrecognized glyph; a drawing system uses an N-best lists to display various ways of grouping similar glyphs.

## 7. Conclusion

The user interface plays a significant role in a diagram recognition system. Unfortunately, it is difficult to define precise goals for a user interface, to find clear guidelines for user interface construction, or to quantify the performance of a user interface. We have reviewed the interaction between diagram editing, generation and recognition and have discussed five categories of user interface issues: quantifying the performance of a user interface, supporting diagram input, executing the recognizer, displaying recognition results, and supporting user correction of recognition errors. Many open problems remain in all of these areas. We look forward to discussions and future work which will lead to improved strategies and criteria for designing, implementing, and evaluating user interfaces for on-line diagram recognition systems.

## Acknowledgments

The work reported in this paper benefited greatly from discussions with and software implementation by Sean Chen, T. C. Nicholas Graham, Jeremy Hussell, Alvin Jugoon, David Tausky, Jeb Thorley, Randall Tuesday, and James Wu. Funding by NSERC (Canada's Natural Sciences and Engineering Research Council) and CITO (Communications and Information Technology Ontario) is gratefully acknowledged.

## References

- [1] J. Arvo, K. Novins, "Smart Text: A Synthesis of Recognition and Morphing," *Proc. AAAI Spring Symposium on Smart Graphics*, Stanford, Cal., March 2000, 140–147.
- [2] S. Bagley, G. Kopec, "Editing Images of Text," *Communications of the ACM*, **37**(4), Dec. 1994, 63–72.
- [3] D. Blostein, L. Haken, "The Lime Music Editor: A Diagram Editor Involving Complex Translations," *Software – Practice and Experience*, **24**(3), March 1994, 289–306.
- [4] D. Blostein, L. Haken, "Using Diagram Generation Software to Improve Diagram Recognition: A Case Study of Music Notation," *IEEE Trans. Pattern Analysis and Machine Intelligence*, **21**(11), Nov. 1999, 1121–1136.
- [5] K. Brooks, "Lilac: A Two-View Document Editor," *IEEE Computer*, June 1991, 7–19.
- [6] M. Brown, "Zeus: A System for Algorithm Animation and Multi-View Editing," *Proc. IEEE Workshop on Visual Languages*, Kobe, Japan, Oct. 1991, 4–9.
- [7] J. Carroll, editor, *Scenario-Based Design: Envisioning Work and Technology in System Development*, John Wiley, 1995.
- [8] HCI journals: *ACM Transactions on Computer-Human Interaction*; *International Journal of Man-Machine Studies*; *Human-Computer Interaction*; *Behavior and Information Technology*; *Human Factors*.
- [9] W. Cushman, P. Ojha, C. Daniels, "Usable OCR: What are the Minimum Performance Requirements?," *Proc. ACM SIGCHI 1990 Conf. Human Factors in Computing Systems*, Seattle, Washington, April 1990, 145–151.

- [10] C. Damm, K. Hansen, M. Thomsen, "Tool Support for Cooperative Object-Oriented Design: Gesture Based Modeling on an Electronic Whiteboard," *Proc. CHI 2000*, The Hague, Netherlands, April 2000, 518–525. See also [www.daimi.au.dk/~knight/](http://www.daimi.au.dk/~knight/).
- [11] D. Diaper, "Task Observation for Human-Computer Interaction," in D. Diaper, editor, *Task Analysis for Human-Computer Interaction*, Ellis Horwood, 1989, 210–237.
- [12] T. C. N. Graham, L. Watts, G. Calvary, J. Coutaz, E. Dubois, L. Nigay, "A Dimension Space for the Design of Interactive Systems Within their Physical Environments," *Proc. Designing Interactive Systems (DIS 2000)*, 2000, 406–416.
- [13] M. Gross, E. Do, "Ambiguous Intentions: a Paper-like Interface for Creative Design," *Proc. Ninth Annual Symposium on User Interface Software and Technology (UIST'96)*, Seattle, Washington, Nov. 1996, 183–192.
- [14] K. Höök, "Designing and Evaluating Intelligent User Interfaces," *Proc. 1998 ACM Int'l Conf. Intelligent User Interfaces*, San Francisco, California, 1998, 5–6.
- [15] R. Hsu, W. Mitchell, "After 400 Years, Print is Still Superior," *Communications of the ACM*, **40**(10), Oct. 1997, 27–28.
- [16] L. Julia, C. Faure, "Pattern Recognition and Beautification for a Pen Based Interface," *Proc. Third Int'l Conf. Document Analysis and Recognition*, Montreal, Canada, August 1995, 58–63.
- [17] E. Lank, J. Thorley, S. Chen, D. Blostein, "On-line Recognition of UML Diagrams," *Proc. Sixth Int'l Conf. Document Analysis and Recognition*, Seattle, Washington, Sept. 2001, 356–360.
- [18] A. Long, J. Landay, L. Rowe, J. Michiels, "Visual Similarity of Pen Gestures," *Proc. CHI 2000 Human Factors in Computing Systems*, The Hague, April 2000, 360–367.
- [19] J. Mankoff, S. Hudson, G. Abowd, "Providing Integrated Toolkit-Level Support for Ambiguity in Recognition-Based Interfaces," *Proc. CHI 2000 Human Factors in Computing Systems*, The Hague, Netherlands, April 2000, 368–375.
- [20] J. McGrenere, W. Ho, "Affordances: Clarifying and Evolving a Concept," *Proc. Graphics Interface 2000*, 179–186.
- [21] B. Myers, "User Interface Software Tools," *ACM Trans. Computer-Human Interaction*, **2**(1), 1995, 64–103.
- [22] M. Nakagawa, K. Machii, N. Kato, T. Souya, "Lazy Recognition as a Principle of Pen Interfaces," *Proc. INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, Amsterdam, 1993, 89–90.
- [23] J. Nielsen, *Usability Engineering*, Academic Press, San Diego, California, 1993.
- [24] J. Nielsen, J. Levy, "Measuring Usability: Preference vs. Performance," *Communications of the ACM*, **37**(4), April 1994, 66–75.
- [25] I. Phillips, A. Chhabra, "Empirical Performance Evaluation of Graphics Recognition Systems," *IEEE Trans. Pattern Analysis and Machine Intelligence*, **21** (9), Sept. 1999, 849–870.
- [26] S. Smithies, K. Novins, J. Arvo, "A Handwriting-Based Equation Editor," *Proc. Graphics Interface '99*, Kingston, Ontario, June 1999, 84–91.
- [27] J. Tang, S. Minneman, "VideoWhiteboard: Video Shadows to Support Remote Collaboration," *Proc. Human Factors in Computing Systems*, 1991, 315–322.
- [28] B. Vander Zanden, S. Venckus, "An Empirical Study of Constraint Usage in Graphical Applications," *Proc UIST'96 – Ninth Annual Symposium on User Interface Software and Technology*, Seattle, Washington, Nov. 1996, 137–146.
- [29] R. Zanibbi, D. Blostein, J. Cordy, "Baseline Structure Analysis of Handwritten Mathematics Notation," *Proc. Sixth Int'l Conf. Document Analysis and Recognition (ICDAR 2001)*, Seattle, Washington, Sept. 2001, 768–773.
- [30] R. Zanibbi, K. Novins, J. Arvo, K. Zanibbi, "Aiding Manipulation of Handwritten Mathematical Expressions through Style-Preserving Morphs," *Proc. Graphics Interface 2001*, Ottawa, Ontario, June 2001, 127–134.