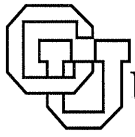
Supporting the End-User Programmer as a Lifelong Learner

Chris DeGiano
Mike Eisenberg

CU-CS-761-95



University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

Supporting the end-user programmer as a lifelong learner

Chris DiGiano
Mike Eisenberg

Department of Computer Science
University of Colorado at Boulder, CB 430
Boulder, CO 80309-0430

Abstract
When end users of applications attempt to learn application-oriented programming languages, they find themselves cast as students in a complex and often daunting educational task. Traditional methods of teaching programming skills to end users are ineffective and overwhelming. A more suitable means of teaching expressive programming skills to end users is suggested by viewing these users as lifelong learners. In this paper we propose a framework for the design of learnable end-user programmable systems, and describe a prototype application, Chart 'n' Art, that embodies this framework.

Introduction
Wisdom is not a product of schooling but the life long attempt to acquire it.
—Albert Einstein

End-user programming offers the opportunity to transcend the built-in functionality of software, empowering those without computer expertise to be more creative and expressive users. Extending or customizing these "programmable applications"(Eisenberg, 1991) can be as simple as making selections from a menu with the mouse, or as complex as composing small programs using a formal written language. This programmability should not, and often cannot, be mastered immediately, and for this reason end-user programming environments typically ensure that "a critical subset of the functionality of the system can be quickly learned and is sufficient for getting useful work done."(Nardi, 1993, p 5)

The person writing programs with these kinds of applications is a unique type of user. First and foremost, end-user programmers are trying to accomplish tasks such as data analysis or chart design which have little, if anything, to do with computers per se. They have become programmers not because of any intrinsic interest in computers, but because the functionality offered by the application they use fails to meet their specific needs. Indeed, many end-user programmers would be reluctant to admit that what they are doing is actually programming.

For programmable systems which provide a formal end-user language—Excel, AutoCAD, Mathematica and HyperCard to name a few—the task of learning the new notation can be daunting, if not prohibitive. In this paper we focus on the learnability of these systems, note the weaknesses of learning support mechanisms commonly encountered by their users, and in response offer new approaches founded on a respect for the user as a lifelong learner which we illustrate with an application named *Chart 'n' Art.*

Learning an end-user programming system
The critical factor in the acceptance and creative use of an end-user language, and, in fact, the central problem for the entire field of programmable applications is learnability. Common learning support mechanisms include training classes, tutorials, and technical support hotlines. In most cases end-user programmable systems come with a printed manual, and many applications such as Excel feature on-line help that acts in part as a language glossary.

Still, a problem with these traditional means of teaching end-user programming is that they make several wrong assumptions about the end-user. Tutorials and training classes expect learners to take in a large amount of information and apply it at some later time to their work. This runs counter to the typical observation that end-users prefer to learn about new features only when they have the time and can see immediate and tangible benefits for their daily activities.(cf. Mackay, 1991) Manuals and on-line help would seem to offer a better alternative to tutorials in that users can extract smaller portions of information as needed, but they too have some problems. These resources expect users to be able map their particular tasks to terminology presented in the text. However, ethnographic studies of end-user programming suggest people would rather turn to experienced coworkers who can make those mappings for them.(cf. Gantt and Nardi, 1992; Nardi and Miller, 1991)

End-user programmers as lifelong learners

Understanding the specific needs and preferences of users who are learning programming is key to determining the most appropriate pedagogical support. We believe that these users can be best understood by considering them as *lifelong learners*. That is, individuals who are building their knowledge of programming throughout the course of their regular interactions with applications. Using this perspective, certain important characteristics of end-users become apparent.

The user is primarily interested in learning things which seem useful at the moment. As a lifelong learner, the end user is in no hurry to learn everything—just the things which can be applied to the current task. In other words the end user is primarily interested in acquiring *situated knowledge*(Winograd and Flores, 1986) of programming.

The user may only want to learn about programming in small increments. End users may accept short, focused learning opportunities but will avoid lengthy digressions from the tasks at hand, even if applicable to their current activity.

The user wants learning to be informal. The prospect of spending a lifetime conforming to a rigid computer-centered curriculum is frightening indeed, especially for someone with little or no interest in computers per se. Rather, end-user programmers would prefer learning opportunities which are unstructured, self-paced and even playful.

The user does not want to be constrained by a less than full-fledged language. Users do not want to find themselves limited by a language prohibiting complete creative and expressive freedom, especially after having invested considerable time and energy learning its basic functionality.

In summary, we can best characterize end-user programmers as lifelong learners of programming who would prefer the opportunity to learn an unconstrained language in ways which are situated, incremental, and informal.

Implications for the design of end-user programmable systems

By considering the end-user programmer as a lifelong learner we can come to a clearer understanding of the appropriate support for learning an application-oriented language. In this section we present Chart 'n' Art (Figure 1) as an example of what the lifelong learning perspective can mean for the design of end-user programmable systems.

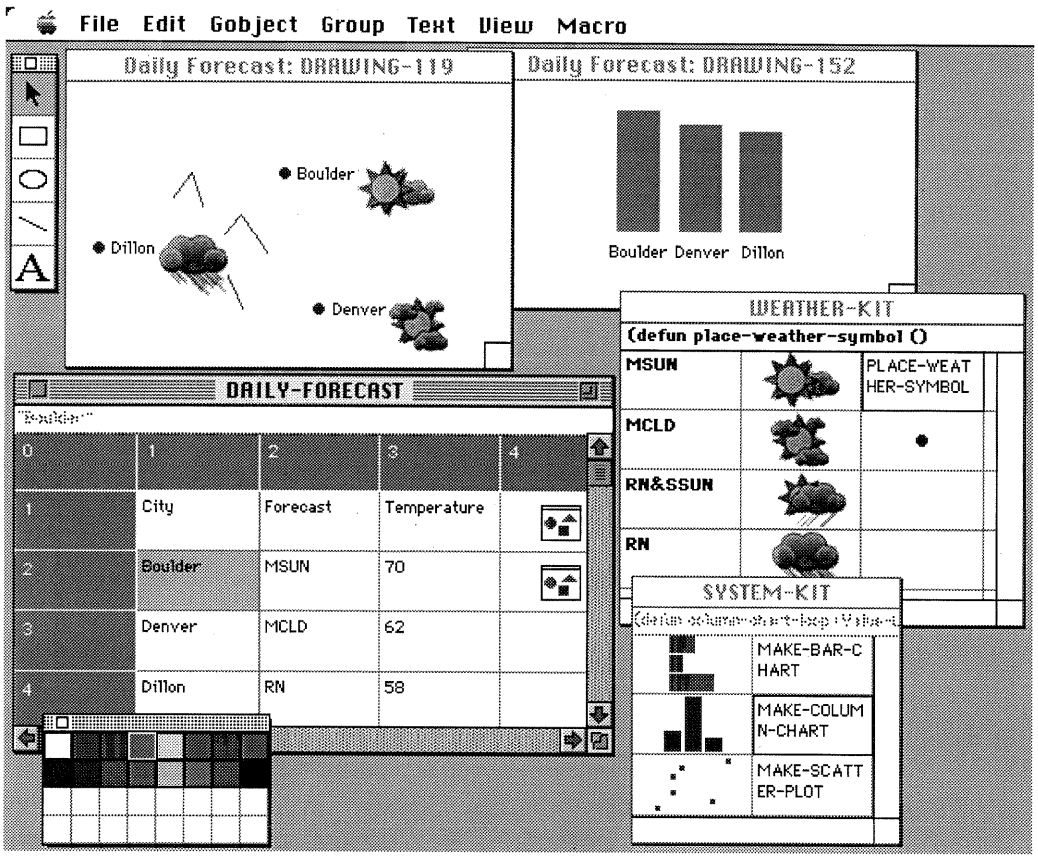


Figure 1

Chart 'n' Art main windows and palettes. The "Daily-Forecast," "Weather-Kit," and "System-Kit" windows hold data, pictures, and procedures for creating the diagrams in the two drawing windows.

Chart 'n' Art is a programmable information visualization tool currently being developed at the University of Colorado which combines the functionality of a spreadsheet and drawing package. The spreadsheet is used to maintain a "kit" of data related to some particular diagram or diagram type; the drawing package is used to construct the actual visualizations. In each portion of the program, direct manipulation may be employed to perform standard operations—for instance, in the drawing package, one can create various shapes and lines via mouse operations in a drawing window (much as in typical graphics applications). Similarly, one can select cells in the spreadsheet (using the mouse button) to enter values or formulas.

Woven through both the spreadsheet and drawing functionality is an end-user language that is an extension of Lisp. The language can be used to create or modify spreadsheet cells, introduce new graphical objects into a drawing, or adjust the attributes of existing graphical elements. Furthermore, Lisp expressions can be used as a computational "adhesive" that connects the (typically disjoint) worlds of spreadsheet and drawing program by mapping spreadsheet data into graphical elements, and vice versa. The language can in fact be used to express iterative and recursive operations succinctly, and allows the user to express diagramming techniques suited to a wide variety of information. Throughout, the system tries to make use of what we call *self-disclosure*. That is, Chart 'n' Art makes available to the user the language expressions behind the actual implementation of direct manipulation operations, menu commands, and other kinds of built-in functionality.

Chart 'n' Art is specifically designed to encourage and support the acquisition of its end-user language by providing situated, incremental, and informal learning mechanisms in a full-fledged programming environment. Let us consider each of these points in turn.

Situated learning. Appropriately situating instruction involves adapting it to the particular environment in which the knowledge will actually be used.(cf. Resnick, 1989). One way this is accomplished in Chart 'n' Art is by accompanying almost every mouse action with a textual entry in a "transcript" window as shown in Figure 2. The transcript entry is a short Lisp expression which could have been used to achieve the same results. Thus, by highlighting the correspondence between interaction modes, users can gradually become familiar with the parts of the language appropriate to their current activity. The transcript window in Figure 2, for example, indicates, among other things, that the user could have called the `set-color` function instead of selecting a color from the palette with the mouse. In designing Chart 'n' Art we have maintained a careful coordination between direct manipulation and language operations to assure each direct manipulation command has an matching linguistic expression or short series of expressions.

The simple transcript window expressions are limited in their pedagogical value to familiarizing the end user to the language's vocabulary, syntax, and some data structures such as for representing coordinates. Teaching end-user programmers how to compose the language into meaningful programs is a more challenging problem. One useful approach is presenting complete example programs. Studies of beginning computer science students suggest working examples are a powerful educational technique.(cf. Anderson, 1987; Pirolli and Anderson, 1985) However, as discussed earlier, the end-user programmer will not tolerate examples presented out of context.

To address this issue Chart 'n' Art builds example-based learning opportunities into common chart creation commands such as for making scatter plots or bar charts. A catalog of these commands called the "System Kit" (see Figure 3) was designed to make prototypical charts and their example code easily accessible. Once users select a chart type with the mouse, they can either press the enter key to generate a graph, or double click to examine its Lisp implementation. In this way users can begin to learn from the programs which implement the commands they regularly apply to their designs, and eventually can base their own custom chart types on these exemplars.

Incremental learning. Viewing end-user programmers as incremental learners creates new challenges for the system designer. Fortunately, such users do not require (and probably do not want) rapid acquisition of programming skills or immediate fluency with an application-oriented language. Thus, learnable programmable applications do not need to promise the kind of dramatic skill acquisition possible through dedicated intelligent tutoring systems such as the Lisp Tutor.(Anderson and Reiser, 1985) For the incremental learner, the system simply has to provide a minimum amount of assistance to make users feel they are not wasting their time by trying to acquire the end-user language. In accommodating this learning style, however, we cannot lump users into fixed categories, but instead must consider their expertise to lie on a continuum. Additionally, instructional information must be organized in very small but effective units which will educate the user and facilitate the current task..

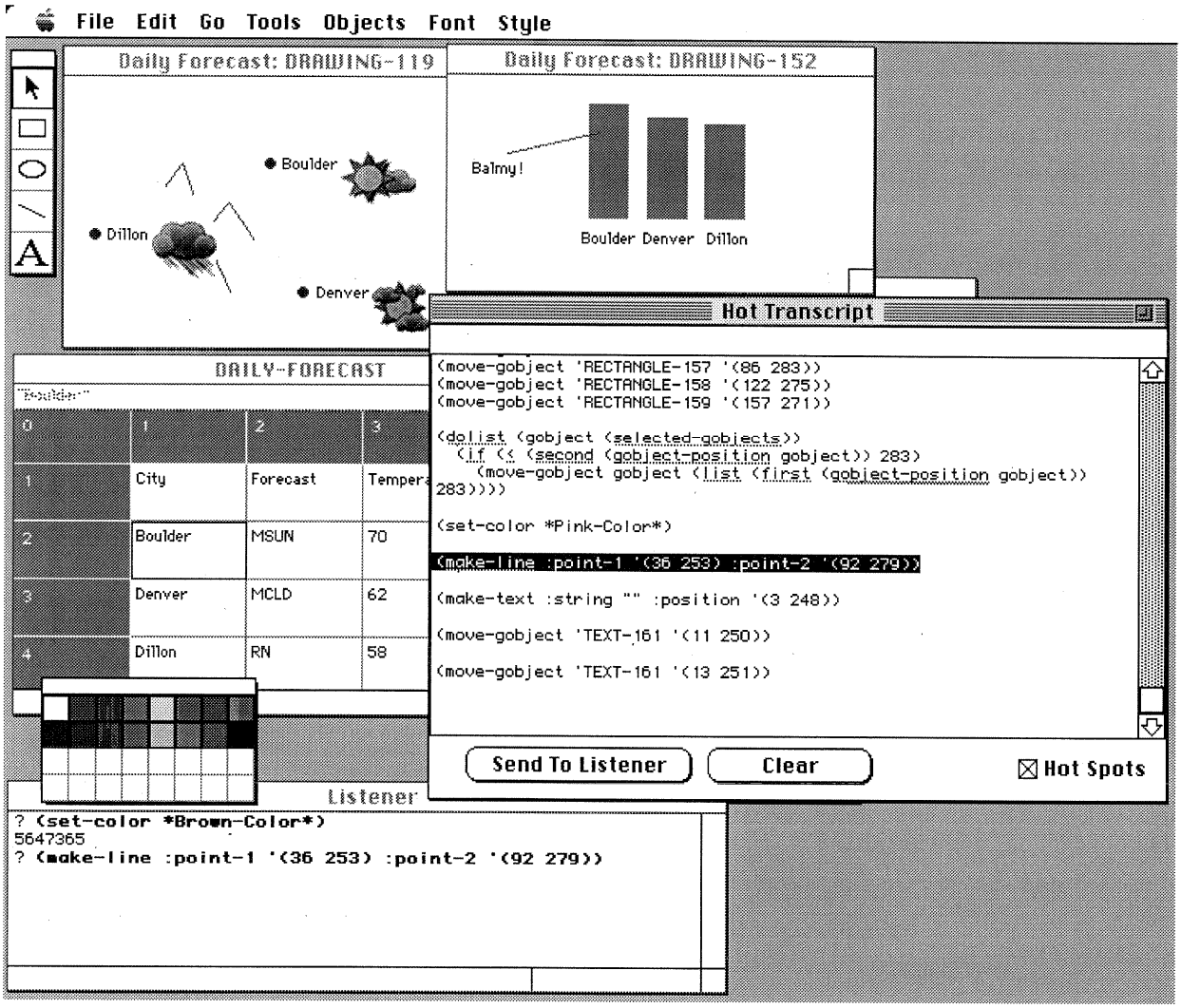


Figure 2

Chart 'n' Art's "hot" transcript window showing linguistic equivalents to direct manipulation operations. Users can click transcript entries for more information. The "Listener" window is where users can experiment with the language

Chart 'n' Art respects the notion of incremental presentation of information by offering transcript messages which are succinct yet informative. From these short language examples appearing in the transcript window, the user is given the opportunity to begin constructing bit by bit a model of the Lisp interpreter and the general schema of the end-user language. Research on learning from examples (cf. Anderson, 1987; Lewis, 1988; Lewis, Hair, and Schoenberg, 1989) suggests users can extrapolate surprising detailed and accurate information about a command language from just a few exemplars. Our own preliminary tests with Chart 'n' Art indicate that non-Lisp programmers can quickly identify the basic open-parenthesis-operation-operand-closed-parenthesis syntax of the language by examining items generated in the transcript window as a result of their direct manipulation operations.

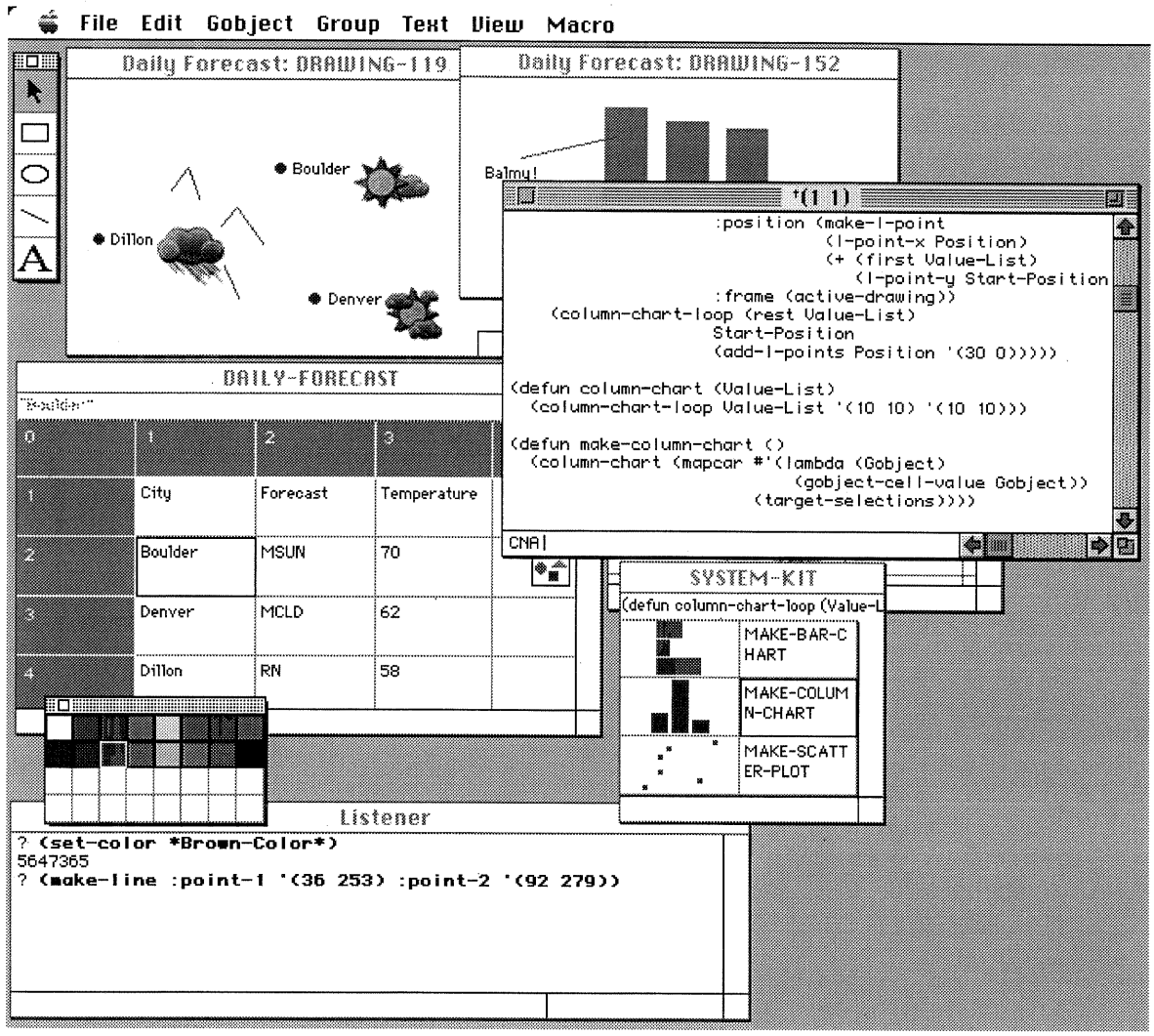


Figure 3

Chart 'n' Art's system kit example program for creating a column chart.

The ability to interleave direct manipulation and linguistic operations in the midst of a single information visualization task is yet another way Chart 'n' Art supports incremental learning. Because users can employ direct manipulation as a crutch, they do not have to be completely comfortable with the end-user language in order to start using it in their activity. For instance, a user may want to write a program which creates a temperature column chart similar to that in Figure 1, but using representative colors. Knowing how to proportionally size rectangles in the language is a good enough start, since the user knows she can select colors from palette with the mouse until she has a better grasp of the Lisp expressions for manipulating color.

Informal learning. Because of users' preference for informal learning, end-user programmable systems should support self-paced exploration of the application-oriented language and facilitate rapid transition between working and learning. Chart 'n' Art allows for rapid exploration by generating hypertext links on the fly for each new transcript entry. Users do not have to look up terms in a reference guide, which would be slow even if on-line. Instead, they click on a word in the transcript window, and related information appears on the screen.

Users can easily experiment with newly encountered Lisp expressions through Chart 'n' Art's "forgiving" language interface. Major linguistic commands are undoable, so that the user can experiment with an expression but then quickly retract its consequences. Undoable linguistic operations provide consistency with direct manipulation commands which are also retractable. More importantly, undoable language creates a non-threatening learning environment which reduces fear by eliminating the "catastrophic consequences of wrong behavior."(Burton, Brown, and Fischer, 1984)

Full-fledged language. Rather than protecting users from the standard constructs of programming languages such as variables, conditionals, and iteration, Chart 'n' Art entrusts them with the responsibility of eventually coming to terms with these concepts. While it is possible to debate what the most appropriate language may be for end-users to learn, we elected to provide a complete Lisp programming environment. Chart 'n' Art itself is implemented in this very same Lisp environment which means end users can, given sufficient time for learning (possibly years), recreate the entire application for themselves. Such expertise is possible, at least theoretically, because although users are initially exposed to only the most basic commands for manipulating graphical objects, spreadsheet cells, and list data structures, the entire repertoire of Lisp expressions is made available.

Related work

The Chart 'n' Art programmable application and the lifelong learning perspective on end-user programmers which has motivated its design are clearly influenced by a significant collection of prior work. Any serious study of end-user programming must acknowledge Gantt and Nardi's priceless observations of both its possibility and rich context.(cf. Gantt and Nardi, 1992; Nardi, 1993; Nardi and Miller, 1990; Nardi and Miller, 1991) Their work clearly supports the notion that users learning programming languages have needs which are not being explicitly addressed in the design of traditional environments. One issue that Gantt and Nardi stress which we do not cover in this paper is that end-user programming is typically a highly collaborative activity between users, "local developers," and professional programmers.

Our work also borrows much from research by Fischer, et al on *domain-oriented design environments*(Fischer, 1992), and, more recently, by Eisenberg and Fischer on *programmable domain-oriented design environments*.(Eisenberg and Fischer, 1994) Their efforts make a strong case for the educational potential of task-specific tools for design. Although Chart 'n' Art currently lacks the critiquing mechanisms often found in such systems, its situated learning mechanisms and catalog are consistent with the domain-oriented design environment architecture.

Finally, our criteria for system design show a close match with Resnick's rethinking of instructional theory based on current cognitive research.(cf. Resnick, 1989) She posits that learning is a process of knowledge construction which is knowledge dependent and is highly situated. She also explores a key assumption in the design of Chart 'n' Art—that people can be self-motivated knowledge discoverers—and concludes that it may be necessary to actively stimulate certain students and teach them knowledge construction strategies.

Conclusions

We have argued that by viewing end-user programmers as lifelong learners we gain an insightful perspective into the design of appropriate learning environments for acquiring application-oriented languages. In particular, we claim that end users will be more willing to learn systems which respect their preference for situated, incremental, and informal learning and their need for a full-fledged language. Detailed user studies of our prototype system Chart 'n' Art will be necessary to determine the validity of our claim.

It is important to note that what is being learned through Chart 'n' Art is not simply Lisp, but more broadly a rich linguistic representation of a particular domain, namely, information visualization. Thus, it is fair to say that our approach has implications towards learning not only programming languages, but complete computational tools for creative and expressive activity.

Acknowledgments

The authors would like to thank the members of the Human-Computer Communications group at the University of Colorado who contributed to the concepts in this paper. We would especially like to thank Clayton Lewis for his thoughts on example-based learning, and Alex Repenning, John Rieman, and Tammy Sumner for their user anecdotes. The research was supported by the National Science Foundation under grants IRI-9015441 and MDR-9253425. The second author is supported in addition by a National Science Foundation Young Investigator grant (IRI-9258684).

References

Anderson, J.R. (1987). Causal analysis and inductive learning. *Proceedings of the Fourth International Machine Learning*, 288-299.

Anderson, J.R., and B.J. Reiser (1985). *The LISP Tutor.*

Burton, R.R., J.S. Brown, and G. Fischer (1984). Analysis of Skiing as a Success Model of Instruction: Manipulating the Learning Environment to Enhance Skill Acquisition. In *Everyday Cognition: Its Development in Social Context.* Edited by J. L. B. Rogoff. 139-150. Cambridge, MA - London: Harvard University Press.

Eisenberg, M. (1991). *Programmable Applications: Interpreter Meets Interface.* 1325.

Eisenberg, M., and G. Fischer (1994). Programmable Design Environments: Integrating End-User Programming with Domain-Oriented Assistance. In *Human Factors in Computing Systems, CHI'94 Conference Proceedings (Boston, MA).* 431-437.

Fischer, G. (1992). Domain-Oriented Design Environments. In *Proceedings of the 7th Annual Knowledge-Based Software Engineering (KBSE-92) Conference (McLean, VA).* 204-213. Los Alamitos, CA: IEEE Computer Society Press.

Gantt, Michelle, and Bonnie A. Nardi (1992). Gardeners and Gurus: Patterns of Cooperation among CAD Users. In *Proceedings of ACM CHI'92 Conference on Human Factors in Computing Systems.* 107-117.

Lewis, C. (1988). Why and How to Learn Why: Analysis-Based Generalization of Procedures. In *Cognitive Science.* 211-256.

Lewis, Clayton, D. Charles Hair, and Victor Schoenberg (1989). Generalization, Consistency, and Control. In *Proceedings of ACM CHI'89 Conference on Human Factors in Computing Systems.* 1-5.

Mackay, Wendy E. (1991). Triggers and Barriers to Customizing Software. In *Proceedings of ACM CHI'91 Conference on Human Factors in Computing Systems.* 153-160.

Nardi, B.A. (1993). *A Small Matter of Programming.* Cambridge, MA: The MIT Press.

Nardi, Bonnie A., and James R. Miller (1990). The Spreadsheet Interface: A Basis for End User Programming. In *Proceedings of IFIP INTERACT'90: Human-Computer Interaction*. 977-983.

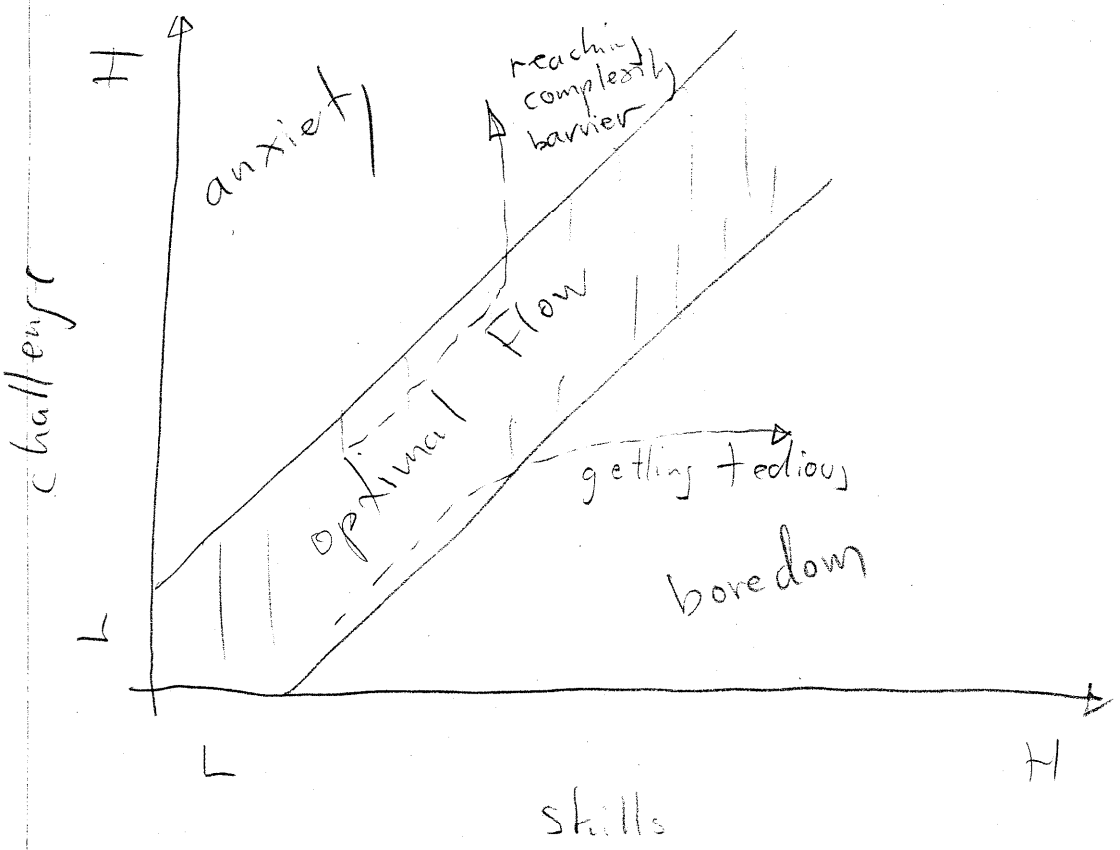
Nardi, Bonnie A., and James R. Miller (1991). Twinkling Lights and Nested Loops: Distributed Problem Solving and Spreadsheet Development. *International Journal of Man-Machine Studies* 34 (2): 161-184.

Pirolli, P.L., and J.R. Anderson (1985). The Role of Learning from Examples in the Acquisition of Recursive Programming Skills. In *Canadian Journal of Psychology*. 240-272.

Resnick, L.B. (1989). Introduction. In *Knowing, Learning, and Instruction: Essays in Honor of Robert Glaser*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Winograd, Terry, and Fernando Flores (1986). *Understanding Computers and Cognition*. Norwood, NJ: Ablex Publishing Corporation.

Optimal Experience



Brigham dimensions:

- Expressiveness
- Facility
- Flexibility