British Computer Society Conference Series 7

People and Computers VIII

Proceedings of HCI 93,
Loughborough, September 1993

Edited by

J.L. Alty
University of Loughborough

D. Diaper
University of Liverpool

S. Guest
University of Loughborough

Published on behalf of
THE BRITISH COMPUTER SOCIETY
by

CAMBRIDGE
UNIVERSITY PRESS

Preface

The Eighth Conference in the *People and Computers* Series was held at Loughborough University of Technology in September 1993. It is one a series of conferences held under the auspices of the British HCI Group (a Specialist Group of the British Computer Society). This volume brings together all the papers accepted for the conference proceedings. From the many high quality papers submitted to the conference, only about one third could be accepted for inclusion after being rigorously refereed by three independent experts in the field. We would like to re-iterate the comment made in the preface last year that, for conferences such as the HCI series, publication selection criteria are often as stringent as for some learned journals.

As usual the content reflects current interests in the Human Computer Interaction research field. Although it is often difficult to characterise papers into streams we have selected the following broad section headings — "User Interface Design", "User Modelling", "Tools or Techniques" (where we have combined two sessions from the conference with similar content). These papers are all concerned in some way with the designing and building of interfaces. Equally important is evaluation, and the sections "Evaluation Issues" and "User Evaluation" are intended to provide a forum for discussion of such issues. The sections "Computer Supported Cooperative Work", "Programming" and "Hypertext", all reflect the widening interest area of HCI research. We hope that authors will forgive us if they feel that their paper is wrongly categorised.

The volume commences with two of the papers presented by our keynote speakers. Eric Hollnagel, who is Principal Scientist with Computer Resources International A/S, is well-known in the field of cognitive ergonomics and intelligent decision support systems. His paper, "The Design of Reliable Human-Computer Interaction: The Hunt for Hidden Assumptions", examines the role of implicit assumptions about human performance which have so characterised early HCI design. The second keynote speaker, Gerhard Fischer, who is from the Institute of Cognitive Science at University of Colorado, Boulder, USA, presents a paper entitled "Beyond Human Computer Interaction: Designing Useful and Usable Computational Environments". He puts the case for pushing human-centred design further forward, and argues for new conceptual and computational environments which will enable the domain specialists to be more independent from computer specialists. He calls for the power of the "high-tech scribes" to be limited just as the role of the 'scribes' of the Middle Ages was redefined. The third keynote speaker at the conference was Karmen Guevara who spoke about "HCI will need to Change because the World is Changing".

This year we had some additional sessions at the conference concerned with "Industrial Applications of HCI" and "Current Research". By their very nature, these sessions were intended to be very up-to-date and therefore had a deadline way beyond the publication deadline for this book.

The Editors would like to thank all those people who gave their help some willingly during the conference and particularly towards the production of these proceedings. In particular we would like to thank Teresa Kennedy and Jo McOuat for their help before, and during, the Conference. Finally we would like to thank all referees and authors for keeping to the very tight deadlines imposed on us by the production schedule. In particular the evaluations of the referees has contributed enormously to the quality of the papers in this volume.

Prof James Alty
Department of Computer Studies, Loughborough University of Technology.

Dr Dan Diaper
Department of Computer Science, University of Liverpool.

Dr Steven Guest
Department of Computer Studies, Loughborough University of Technology.

Beyond Human Computer Interaction: Designing Useful and Usable Computational Environments

Gerhard Fischer

Department of Computer Science and Institute of Cognitive Science, University of Colorado at Boulder, Boulder, CO 80309 0430, USA.
EMail: *gerhard@cs.colorado.edu*

Human–computer interaction has refocussed many research efforts within computer science from a technology-centered view to a human- centered view. But current research efforts and systems (both prototypes and those commercially available) are just the beginning rather than the end. Conceptual frameworks and computational environments are needed that will give domain workers more independence from computer specialists. Just as the pen was taken out of the hands of the scribes in the middle ages, the power of high-tech computer scribes should be re-defined. To turn computers into convivial tools requires that end users themselves can use, change and enhance their tools and build new ones without having to become professional-level programmers.

This article explores a number of future themes transcending current views of human–computer interaction. It describes domain-oriented design environments as new prototypes of computational environments which are simultaneously useful and usable by focusing on humans and their tasks.

Keywords: human–computer interface design, cooperative work, domain oriented design.

1. Introduction

Each of us might have a different opinion about what the most important problems in the unexplored territory beyond current research in human–computer interaction (HCI) are — for other opinions see (Carroll, 1993; Kay, 1990; Lewis, 1990; Norman, 1990). I will enumerate and justify some of the views characterizing the general conceptual framework for our own current research efforts centered around arguments, claims, and hypotheses that current research in human–computer interaction is often based on misconceptions of

what the essential problems and issues are. My contribution will address the following misconceptions:

- The user interface is the major (and maybe only) problem for human–computer interaction research.
- Computer systems should be usable — ignoring the fact that usable systems that are not also useful are of no value.
- Most users are interested in computers per se, rather than in their tasks.
- Access to computers should be restricted to high-tech scribes (i.e. trained computer specialists).
- Design can ignore the traditions of the user community that the system will serve.

These 'misconceptions' will be discussed and alternative views will be presented. I will briefly discuss how our own research about domain-oriented design environments tries to explore some of these alternatives and challenges for future human–computer interaction research.

2. Human–Computer Interaction Is More than User Interfaces

Human–computer interaction is more than 'screen-deep' (Laurel, 1991). The interface is important — but if we change only interfaces and not the systems behind them we will only be able to scratch the surface. We should strive for 'interfaceless systems' in which nothing stands between users and their tasks (and in which system objects become 'ready-at-hand' in a Heideggerian sense). Human–computer interaction should be concerned with tasks, with shared understanding, with explanations, justifications, and argumentation about actions, and not just with interfaces. According to Kay:

> "Many are just discovering that user interface design is not a sandwich spread - applying the MacIntosh style to poorly designed applications and machines is like trying to put Bearnaise sauce on a hotdog!" (Kay, 1990)

In a similar way, Norman argues:

> "The real problem with the interface is that it is an interface. Interfaces get into the way. I don't want to focus my energies on an interface. I want to focus on the job." (Norman, 1990)

Although the usual concerns of interface designers (creating more legible types, designing better scroll bars, integrating color, sound and voice, developing models of keystroke use (Card, Moran & Newell, 1983)) are all important, they are secondary considerations. The essential challenges are improving the way people can use computers to work, think, communicate, learn, critique, explain, argue, debate, observe, decide, calculate, simulate, and design. The emphasis in the future has to be on humans and their tasks — not on computers and their tools.

3. Make Systems Useful and Usable

Useful computers that are not usable are of little help; but so are usable computers that are not useful (Fischer, 1987). One of the major goals of human–computer interaction

research has to be to achieve these two goals — usefulness and usability — simultaneously by breaking the 'conservation law of complexity' (Simon, 1981), which claims that the relationship between the complexity and usability of a system is a given constant. Complexity can be reduced:

1. by exploiting information that is already known and familiar;
2. by using familiar representations (based on previous knowledge and analogous situations);
3. by exploiting the strengths of human information processing;
4. by integrating knowledge in the head with knowledge in the world; and
5. by designing 'better' systems that take advantage of the unique possibilities of interactive computer systems.

Computer systems of today primarily model parts of the world and do not just implement algorithms — and the reality of the world is not user-friendly. Systems that try to capture and model reality will therefore be complex, high-functionality systems. High-functionality systems are a consequence of creating knowledge markets (Stefik, 1986) and layered architectures (Dawkins, 1987; Fischer & Girgensohn, 1990), which make it possible to build complex systems that would be infeasible if everything had to be built from scratch. LISP machines and UNIX systems are examples of high-functionality computer systems. LISP machines, for example, offer approximately 30,000 functions and 3,000 object-oriented classes documented on 4,500 pages of manuals. They contain software objects that form substrates for many kinds of tasks. Systems with such a rich functionality offer power but also problems for designers and users. Even experts are unable to master all the facilities of high-functionality computer systems (Draper, 1984). Designers using these systems can no longer be experts with respect to all existing tools — especially in a dynamic environment in which new tools are being added continuously (Eisenberg, 1991). High-functionality computer systems create a 'tool-mastery' burden (Brooks, 1987) that can outweigh the advantage of the broad functionality offered.

Many approaches that represented major advances in human–computer interaction, such as direct manipulation (Hutchins, Hollan & Norman, 1986) (bridging the interface gulf by representing the world of the computer as a collection of objects that are directly analogous to objects in the real world; the Macintosh desktop being the most successful example of this approach), lose some of their power in high-functionality systems in which the complex and abundant functionality can neither be represented explicitly on the screen nor be explored by browsing mechanisms.

4. A Broader View of Communication and Coordination Processes

Most of human–computer interaction research until now has been focused around a single user interacting with a single computer system offering tools rather than task support. Communication and coordination processes can provide a focus for a number of needed research efforts. A communication and coordination perspective illustrates the requirement to include support for communication with:

- ourselves — e.g. capturing our thoughts of the past, allowing us to create personalized information environments that extend the knowledge we can keep in our head (Bush, 1945; Norman, 1993);

- our tools — e.g. knowing which tools exist, how they can be used, and how they can be tailored to our specific needs (Fischer, 1987);
- our colleagues — e.g. supporting long-term, indirect collaboration (Fischer et al., 1992);
- other humans — e.g. supporting interdisciplinary computer-supported cooperative work (Greif, 1988); and
- our agents and critics — e.g. in the context of cooperative problem-solving systems (Fischer et al., 1993).

The following broad classes of communication and coordination processes need to be analyzed, studied, and further supported:

- Communication processes between designers and clients, which create the following challenges:
    a. clients do not know what they want; and
    b. designers and clients need shared knowledge and artifacts for mutual understanding, thus requiring 'languages of doing' (Ehn, 1988) instead of formal representations.
- Communication processes within design teams, because most real tasks are not done by individuals but by groups of people. Members within such teams might have very different interests (for example, waterfall models in software design are heaven for managers and hell for creative programmers).
- Communication processes between designer(s) and knowledge-based design environments in which these environments serve as group and design artifact memories that can be used to support indirect, long-term communication, requiring that discussions about the design must be embedded in the design (Fischer et al., 1992; Trigg, Suchman & Halasz, 1986).

5. Support Human Problem-Domain Interaction

To bring tasks to the forefront, computers must become 'invisible'. To achieve this goal, human–computer interaction needs to advance to *human problem-domain interaction* (Fischer & Lemke, 1988), requiring that the major abstractions of a given domain are modeled in the computer. This will enable users to describe things briefly because the systems understand domain-oriented concepts. To achieve human problem-domain interaction, we have to sacrifice generality for the power of specialized interactions. This domain-oriented design of artifacts supports the grounding of interaction, creates languages of doing, and allows referential anchoring.

Human problem-domain interaction puts owners in charge, by allowing them to communicate with the systems at a level that is *situated* within their own world (Suchman, 1987). By supporting languages of doing (Ehn, 1988) such as prototypes, mock-ups, scenarios, images, or visions of the future, human problem-domain interaction makes it easier for the owners of problems to participate in the design process because the representations of the evolving artifacts are less abstract and less alienated from practical use situations. By keeping owners in the loop, domain-oriented design environments support the integration of problem framing and problem solving (Rittel, 1984; Schoen, 1983); and allow software systems to

deal with fluctuating and evolving requirements. By making information, relevant to the task at hand (Fischer & Nakakoji, 1992), they are able to deliver the right knowledge, in the context of a problem or a service, at the right moment for a human professional to consider.

Achieving the goal of putting problem owners in charge by developing design environments is not only a technical problem, but a considerable social effort. If the most important role for computation in the future is to provide people with a powerful medium for expression, then the medium should support them in working on the task, rather than require them to focus their intellectual resources on the medium itself.

The analogy to writing and its historical development suggests the goal "to take the control of computational media out of the hands of high-tech scribes". Pournelle (1990, p.281 & p.304) argues that "putting owners of problems in charge" has not always been the research direction of the professional computer scientist:

> "In Jesus' time, those who could read and write were in a different caste from those who could not. Nowadays, the high priesthood tries to take over the computing business. One of the biggest obstacles to the future of computing is C. C is the last attempt of the high priesthood to control the computing business. It's like the Scribes and the Pharisees who did not want the masses to learn how to read and write."

6. Redefine the Role of High-Tech Computer Scribes

Domain workers should gain more independence from computer specialists. Just as the pen was taken out of the hands of the scribes in the Middle Ages, the power of the high-tech computer scribes should be re-defined. To turn computers into convivial tools (Illich, 1973) requires that the end users themselves can change their tools and build new ones without having to become professional-level programmers. This implies that not only can they access materials and tools created by others, but they can generate modified and new materials and tools for themselves and for others.

Domain workers are not 'novice' or 'naive' users. They are people who have computational needs and want to make serious use of computers but are not interested in becoming professional programmers. They are skilled and knowledgeable in their respective domains; they use computers by choice, and over extended periods of time. To understand their use of computers requires a new orientation of many current HCI efforts. Rather than focusing on short-term events (i.e. being concerned with events taking between 0.1 and 10 seconds (Carroll & Campbell, 1986)), the educational, social and organizational needs (being concerned with activities that range for days, months, and years (Newell & Card, 1985)) have to be investigated. New themes such as the integration of working and learning, learning on demand (Fischer, 1991), production paradox (Carroll & Rosson, 1987), intrinsic motivation, and possibilities for self-expressions should be assessed. A few of the major efforts to empower end-users and domain workers will be briefly described.

6.1. Computational Environments for Children

One of the earliest efforts to empower end-users was the creation of computational environments for children, such as LOGO (with the embedding of turtle geometry) (Boecker,

Eden & Fischer, 1991; Papert, 1980) and Smalltalk (with the vision of a DYNABOOK behind it) (Kay, 1977). These efforts showed that humans (even children) could use computers to achieve their own goals without requiring detailed knowledge of low-level computational concepts.

6.2. End-User Programming

Professional programmers and end-users define the endpoints of a continuum of computer users. The former like computers because they can program, and the latter because they get their work done. The key to end-user programming is:

1. to offer task-specific languages (Fischer & Lemke, 1988) that take advantage of existing user knowledge (e.g. a mathematician already knows the mathematical knowledge embedded in Mathematica, and an accountant already knows the conceptual model behind spreadsheets);
2. to provide a programming environment that makes the functionality of the system transparent and accessible so that the computational drudgery required of the user can be substantially reduced; and
3. to hide low-level computational details as much as possible from the users. The challenge in developing these environments is (as argued before) to make them useful and usable.

6.3. Programmable Applications

Direct manipulation interfaces have made an important contribution in making computational environments accessible to a large number of users, but they face two important hurdles:

1. the semantics of clicking, dragging, and selection are too impoverished to accommodate the imagination of long-term users; and
2. the specific needs of users cannot be fully anticipated by the original designer. Programmable applications (Eisenberg, 1991) have as their goal the integration of the successes of extensive, learnable direct manipulation interfaces with the rich expressive range of programming languages.

6.4. Communities of System Users

The high-functionality systems mentioned earlier have as a consequence that there are *no experts* (i.e. individuals who know everything about a system) any more. For these types of systems, groups, rather than individuals need to be the locus of knowing. In using powerful computational environments in groups, a continuum of users between developers and end-users emerges. Intermediate users are called *local developers* (Nardi Gantt, 1992) or *power-users*. Many systems offer embedded programming environments (often as macro languages or simplified programming languages, such as HyperTalk in HyperCard, AutoLISP in AutoCAD, EmacsLisp in Emacs, extensible databases in Scribe, among others). Local developers create extensions to computational environments either on their own initiatives or upon request from other users, and the whole community of users profits from their efforts.

7. Domain-Oriented Design Environments

Over the last 10 years, our research efforts have tried to take the preceding discussion into account and to develop prototypes of new kinds of computational environments that

allow us to put our evolving conceptual framework to work and test its viability and its shortcomings. Our current prototypes are *domain-oriented design environments* (Fischer, 1992) and we and others have developed numerous prototypes in different domains e.g. floorplan design for kitchens (Fischer & Nakakoji, 1992), user interface design (Lemke & Fischer, 1990), decision support system for water management (Lemke & Gance, 1991), computer network design (Fischer et al., 1992), voice dialog design (Repenning & Sumner, 1992), COBOL programming (Atwood et al., 1991), and lunar habitat design (Stahl, 1993).

The essential components of domain-oriented design environments are – for details see (Fischer, 1992):

1. a construction kit;
2. an argumentative hypermedia system;
3. a catalog of pre-stored designs;
4. a specification component; and
5. a simulation component.

Design environments derive their power from the integration of their components. Used individually, the components are unable to achieve their full potential. Used in combination, each component augments the values of the others, forming a synergistic whole. The integration among the components are supported by various mechanisms (Fischer et al., 1991b):

- *Construction-analyzer* is a critiquing system (Fischer et al., 1991a) that provides access to relevant information in the argumentative issue base. The firing of a critic signals a breakdown to users and provides them with an entry into the exact place in the argumentative hypermedia system where the corresponding argumentation is located.
- *Argumentation-illustrator.* The explanation given in argumentation is often highly abstract and very conceptual. Concrete design examples that match the explanation help users to understand the concept. The *Argumentation-illustrator* (Fischer et al., 1991b) helps users to understand the information given in the argumentative hypermedia by finding a catalog example that illustrates the concept.
- *Catalog-explorer* helps users to search the catalog space according to the task at hand (Fischer & Nakakoji, 1992; Nakakoji, 1993). It retrieves design examples similar to the current partial construction situation and orders a set of examples by their appropriateness to the current partial specification.

8. Moving Beyond Human–Computer Interaction with Domain-Oriented Design Environments

Figure 1 establishes a mapping between the conceptual framework outlined in this paper and the features of design environments.

8.1. Saying the 'Right' Thing at the 'Right' Time in the 'Right' Way

Making information relevant to the task at hand poses many challenges for the design of interactive computer systems, particularly for problems in which the need for information is critical and yet precise information needs cannot be known in advance of attempts to solve the problem. Designers are often unwilling to disrupt the design process to search

Conceptual Issues	Domain-oriented Design Issues
human–computer interaction is more than user interfaces	domain modelling
make systems useful and usable	provide functionality and mechanisms to make information relevant to the task at hand
a broader view of communication and coordination processes between: (1) designer and client; (2) design teams; and (3) human and computer	support for: (1) languages of doing; (2) computer-supported cooperative work; and (3) cooperative problem solving
artifacts do not speak for themselves	critics and simulation components
human problem-domain interaction	represent and present abstraction; make the computer visible
redefine the role of the high-tech computer scribes	empower the owners of problems by allowing them to interact directly with computational environments
support design as tradition and transcendence	task-orientation, critics, seeds, evolutionary growth, and end-user modifiability

Figure 1: Exploration of relevant issues in domain-oriented design environments

for information in large information spaces, even if they know the information exists. In addition, designers may not know when they need information Embedded critics (Fischer et al., 1993) save designers the trouble of explicitly querying the system for information. Critics notify designers of situations indicating the need to reflect (breakdowns) and provide access to information fuelling reflection. The context of the breakdown situation serves as an implicit query that enables embedded critics to deliver relevant information. Designers benefit from needed information without having to explicitly ask for it.

Embedded critics can *deliver* relevant information (Nakakoji, 1993) about which designers were unaware. Critics provide the designer with a pointer into part of the system's information space with which the designer needs to become aware. The designer can further browse the unfamiliar portion of the information space starting from the entry point provided by the critic. Critics afford learning on demand (Fischer, 1991) by letting designers access new knowledge in the context of actual problem situations. They inform users:

1. when they are getting into trouble;
2. when they are missing important information; and
3. when they come up with problematic solutions.

8.2. End-User Modifiability

Design in real world situations deals with complex, unique, uncertain, conflicted, unstable situations of practice. Design knowledge as embedded in design environments will never be complete because design knowledge is tacit i.e. competent practitioners know

more than they can say — (Polanyi, 1966) — and additional knowledge is triggered and activated by situations and breakdowns. These observations require computational mechanisms in support of end-user modifiability (Fischer & Girgensohn, 1990). The *end-user modifiability* of JANUS (Girgensohn, 1992) allows users to introduce new design objects (e.g. a microwave), new critiquing rules (e.g. appliances should be against a wall unless one deals with an island kitchen), and kitchen designs which fit the needs of a blind person or a person in a wheelchair.

Our work on creating domain-oriented design environments has indicated that a promising model for creating them involves the creation of a *seed* for such an environment (Fischer et al., 1992). A seed will be created in cooperation between knowledge engineers and domain experts. It will evolve in response to its extensive use in new design projects in this domain. 'Use' is not just use because requirements fluctuate, change is ubiquitous, and because design knowledge is tacit, design environments need to evolve (Henderson & Kyng, 1991). This evolution is primarily driven by using the existing environment to develop new designs that uncover its limitations through 'breakdowns" (Winograd & Flores, 1986). But these breakdowns are perceived by the people who use the programs, not by the professionals who have developed them in the first place. Supporting end-user modifiability is critical for the evolution of the design environment itself as well as for individual design projects developed within the design environment.

8.3. New Role Distributions: Between High-Tech Scribes and Domain Workers in Domain-Oriented Design Environments

Domain-oriented design environments represent the next step in the historical efforts to make computers invisible behind domain abstractions, thereby allowing users to focus on understanding problems rather than fighting media.

A consequence of domain-oriented design environments is that the professions using computers become further specialized (see Figure 2): high-tech scribes (e.g. knowledge engineers, programmers) in collaboration with domain workers create design environments (at least the seeds for them (Fischer et al., 1992), and domain workers use and evolve the seeded environments.

Domain-oriented design environments reduce the dependency on high-tech scribes by supporting design in use (Henderson & Kyng, 1991), tailorability and customizability (MacLean et al., 1990; Trigg, Moran & Halasz, 1987), and end-user modifiability (Fischer & Girgensohn, 1990; Girgensohn, 1992). They contribute to the goal of convivial computing by resolving the conflict between the generality, power, and rich functionality of modern computer systems and the limited time and effort that domain specialists are willing to spend in solving their problems. They are promising architectures to put owners of problems in charge. They are based on:

1. the basic belief that humans enjoy deciding and doing; and
2. the assumption that the experience of having participated in a problem makes a difference to those who are affected by the solution. People are more likely to like a solution if they have been involved in its generation, even though it might not make sense otherwise.

Figure 3 characterizes the role distribution between high-tech scribes and domain workers in future computing environments.

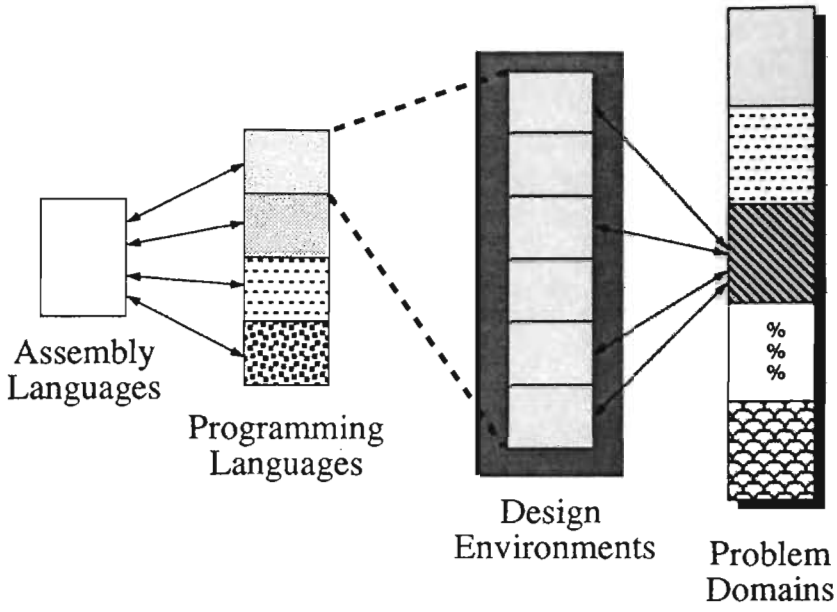

Figure 2: Domain-oriented design environments (In the 1950's programmers had to map problems directly to assembly languages and the assembly programs retained basically no semnatics of the problems to be solved. In the 1960's, general purpose high-level programming languages reduced the transformation distance, the programs written in them were able to retain some problem semantics and the programming profession was specialized into programmers who wrote compilers and programmers who developed programs in high-level languages. Design environments introduce another layer which is domain-oriented. The professions are further specialized into knowledgeengineers who create (in cooperation with domain workers) the seeds for design environments and domain workers who solve problems by exploiting the resources of the design environments. Support for end-user modifiability allows domain workers to extend the functionality of the design environment over time.)

9. Design: Tradition and Transcendence

Successful design has to achieve the two goals of tradition and transcendence simultaneously. Taking tradition into account implies a work-oriented design of artifacts (Ehn, 1988). Tradition can be captured within design environments by supporting domain-oriented abstractions and by reminding users with the help of critics of established design principles.

9.1. Success Models

Because an over-emphasis on tradition often overlooks the fact that new tools change tasks (e.g. electronic forms and how humans can deal with them do not need to be restricted to an imitation of forms on paper), transcending established work practices requires that we have to move 'beyond the city walls'. We have to:

1. Look for success models in other disciplines (e.g. architecture (Cross, 1984), drama (Laurel, 1991), skiing (Burton, Brown & Fischer, 1984), etc.).
2. Forget the vividness of the present and to get back to basic principles that are centered around humans, and to exploit the power of computational media beyond the limitation of paper and desk top metaphors.

High-tech Scribes		**Domain Workers**
Design of generic tools suited for widespread distribution		Design of personal tools and applications
Design of software related to computing as a subject domain		Using programming to incrementally understand ill-defined problems
Development of technical knowledge ('how')	← Creation of seeds →	Development of conceptual knowledge ('what')
Validating and verifying the seed at the formal level (enhancement at the technical level)		Enhancements to the seed in response to breakdowns (enhancements at the conceptual level)
Interpretation, extension, and reconceptualization of technical knowledge ('how')	← Reseeding →	Interpretation, extension, and reconceptualization of conceptual knowledge ('what')

Figure 3: Role distribution between high-tech scribes and domain workers in future computing environments

We have to be aware of the premature establishment of restricted work practices. The HCI community runs the danger that the Macintosh becomes the QWERTY keyboard, COBOL or MS-DOS of user interface design (Kay, 1990).

10. Beyond Technological Aspects

Moving beyond human computer interaction is not only a technical problem, but a considerable social effort. Many HCI approaches (e.g. recording of design rationale, design for reuse) have failed not for their lack of technical sophistication, but by not paying enough attention to human-centered principles — e.g. "who is the beneficiary and who has to do the work?" (Grudin, 1988).

If the most important role for computation in the future is to provide people with a powerful medium for expression, then the medium should support them in working on the task, rather than requiring them to focus their intellectual resources on the medium itself. The analogy to writing and its historical development suggests the goal "to take the control of computational media out of the hands of high-tech scribes". Computational media may turn out to be of greater importance to people than writing because the objects created with them can be interpreted not only by humans (as the printed word) but in part by computers.

Convivial tools and systems — as defined by Illich (1973) — allow users "to invest the world with their meaning, to enrich the environment with the fruits of their vision and to use them for the accomplishment of a purpose *they have chosen*". Conviviality is a dimension that sets computers apart from other communication and information technologies (e.g. television, video disks, interactive videotex) that are passive and cannot conform to the users' own tastes and tasks. Passive technologies offer some selective power, but they cannot be extended in ways that the designer of those systems did not directly foresee. An old Asian Proverb states:

> "Give human beings a fish and they have food for a day — teach them fishing, and they will have food for their whole life."

This can be taken even a step further: if we can provide human beings with the knowledge, the know-how, and the tools for making a fishing rod, they can feed a whole community. The real goal of future computational environments is to provide means to create, criticize and disseminate knowledge and put it to work to assist us in solving our problems and satisfying our needs — and to do so in a way, that these environments are simultaneously useful *and* usable.

Acknowledgments

The author would like to thank the members of the Human–Computer Communication group at the University of Colorado, who contributed to the conceptual framework and the systems discussed in this article. The research was supported by the National Science Foundation under grants No.IRI-9015441 and No.MDR-9253425, and by grants from the Intelligent Interfaces Group at NYNEX and from Software Research Associates (SRA).

References

Atwood, M E, Burns, B, Gray, W D, Morch, A I, Radlinski, E R & Turner, A (1991), "The Grace Integrated Learning Environment — A Progress Report", in *Proceedings of the Fourth International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems (IEA/AIE 91)*, ACM Press, pp.741–745.

Boecker, H-D, Eden, H & Fischer, G (1991), *Interactive Problem Solving Using Logo*, Lawrence Erlbaum Associates.

Brooks, F P (1987), "No Silver Bullet: Essence and Accidents of Software Engineering", *IEEE Computer* 20 (4), pp.10–19.

Burton, R R, Brown, J S & Fischer, G (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*, B Rogoff & J Lave [eds.], Harvard University Press, pp.139–150.

Bush, V (1945), "As We May Think", *Atlantic Monthly* 176 (7), pp.101–108.

Card, S K, Moran, T P & Newell, A (1983), *The Psychology of Human–Computer Interaction*, Lawrence Erlbaum Associates.

Carroll, J M (1993), "Creating a Design Science of Human–Computer Interaction", *Interacting with Computers* 5 (1), pp.3–12.

Carroll, J M & Campbell, R L (1986), "Softening Up Hard Science: Reply to Newell and Card", *Human–Computer Interaction* 2, pp.227–249.

Carroll, J M & Rosson, M B (1987), "Paradox of the Active User", in *Interfacing Thought: Cognitive Aspects of Human–Computer Interaction*, J M Carroll [ed.], MIT Press, pp.80–111.

Cross, N [ed.] (1984), *Developments in Design Methodology*, John Wiley & Sons Ltd.

Dawkins, R (1987), *The Blind Watchmaker*, W W Norton and Company.

Draper, S W (1984), "The Nature of Expertise in UNIX", in *Proceedings of INTERACT'84 — First IFIP Conference on Human–Computer Interaction*, B Shackel [ed.], Elsevier Science (North Holland), pp.182–186.

Ehn, P (1988), *Work-Oriented Design of Computer Artifacts*, Almquist & Wiksell International.

Eisenberg, M (1991), "Programmable Applications: Interpreter Meets Interface", Department of Electrical Engineering and Computer Science, MIT, Technical Report 1325.

Fischer, G (1987), "Making Computers more Useful and more Usable", in *Proceedings of the 2nd International Conference on Human–Computer Interaction*, Elsevier Science (North Holland), pp.97–104.

Fischer, G (1991), "Supporting Learning on Demand with Design Environments", in *Proceedings of the International Conference on the Learning Sciences*, pp.165–172.

Fischer, G (1992), "Domain-oriented Design Environments", in *Proceedings of the 7th Annual Knowledge-Based Software Engineering (KBSE-92) Conference*, IEEE Computer Society Press, pp.204–213.

Fischer, G & Girgensohn, A (1990), "End-User Modifiability in Design Environments", in *Proceedings of CHI'90: Human Factors in Computing Systems*, J C Chew & J Whiteside [eds.], ACM Press, pp.183–191.

Fischer, G, Grudin, J, Lemke, A C, McCall, R, Ostwald, J, Reeves, B N & Shipman, F (1992), "Supporting Indirect, Collaborative Design with Integrated Knowledge-Based Design Environments", *Human–Computer Interaction* 7 (3), pp.281–314.

Fischer, G & Lemke, A (1988), "Construction Kits and Design Environments: Steps Toward Human Problem-domain Communication", *Human–Computer Interaction* 3 (3), pp.179–222.

Fischer, G, Lemke, A C, Mastaglio, T & Morch, A (1991a), "The Role of Critiquing in Cooperative Problem Solving", *ACM Transactions on Office Information Systems* 9 (2), pp.123–151.

Fischer, G, Lemke, A C, McCall, R & Morch, A (1991b), "Making Argumentation Serve Design", *Human–Computer Interaction* 6 (3-4), pp.393–419.

Fischer, G & Nakakoji, K (1992), "Beyond the Macho Approach of Artificial Intelligence: Empower Human Designers — Do Not Replace Them", *Knowledge-Based Systems Journal* 5 (1), pp.15–30.

Fischer, G, Nakakoji, K, Ostwald, J, Stahl, G & Sumner, T (1993), "Embedding Computer-based Critics in the Contexts of Design", in *Proceedings of INTERCHI'93*, ACM Press, pp.157–164.

Girgensohn, A (1992), "End-User Modifiability in Knowledge-Based Design Environments", Department of Computer Science, University of Colorado, Unpublished PhD Dissertation, Also available as TechReport CU-CS-595-92.

Greif, I [ed.] (1988), *Computer-Supported Cooperative Work: A Book of Readings*, Morgan Kaufmann.

Grudin, J (1988), "Why CSCW Applications Fail: Problems in the Design and Evaluation of Organizational Interfaces", in *Proceedings of CSCW'88: Conference on Computer Supported Cooperative Work*, D Tartar [ed.], ACM Press.

Henderson, A & Kyng, M (1991), "There's No Place Like Home: Continuing Design in Use", in *Design at Work: Cooperative Design of Computer Systems*, J Greenbaum & M Kyng [eds.], Lawrence Erlbaum Associates, pp.219–240.

Hutchins, E L, Hollan, J D & Norman, D A (1986), "Direct Manipulation Interfaces", in *User Centered Systems Design: New Perspectives on Human–Computer Interaction*, D A Norman & S W Draper [eds.], Lawrence Erlbaum Associates, pp.87–124.

Illich, I (1973), *Tools for Conviviality*, Harper & Row.

Kay, A C (1977), "Microelectronics and the Personal Computer", *Scientific American*, pp.231–244.

Kay, A C (1990), "User Interface: A Personal View", in *The Art of Human–Computer Interface Design*, B Laurel [ed.], Addison Wesley, pp.191–207.

Laurel, B (1991), *Computers as Theatre*, Addison Wesley.

Lemke, A C & Fischer, G (1990), "A Cooperative Problem Solving System for User Interface Design", in *Proceedings of AAAI-90 (Eighth National Conference on Artificial Intelligence)*, AAAI Press/MIT Press, pp.479–484.

Lemke, A C & Gance, S (1991), "End-User Modifiability in a Water Management Application", Department of Computer Science, University of Colorado, Technical Report CU-CS-541-91.

Lewis, C H (1990), "A Research Agenda for the Nineties in Human–Computer Interaction", *Human–Computer Interaction* 5, pp.125–143.

MacLean, A, Carter, K, Lovstrand, L & Moran, T (1990), "User-Tailorable Systems Pressing the Issues with Buttons", in *Proceedings of CHI'90: Human Factors in Computing Systems*, J C Chew & J Whiteside [eds.], ACM Press, pp.175–182.

Nakakoji, K (1993), "Increasing Shared Understanding of a Design Task between Designers and Design Environments: The Role of a Specification Component", Department of Computer Science, University of Colorado, Unpublished PhD Dissertation, Also available as TechReport CU-CS-651-93.

Nardi Gantt, B A (1992), "Gardeners and Gurus: Patterns of Cooperation Among CAD Users", in *Proceedings of CHI'92: Human Factors in Computing Systems*, P Bauersfeld, J Bennett & G Lynch [eds.], ACM Press, pp.107–117.

Newell, A & Card, S K (1985), "The Prospects for Psychological Science in Human–Computer Interaction", *Human–Computer Interaction* 1 (3), pp.209–242.

Norman, D A (1990), "Why Interfaces Don't Work", in *The Art of Human–Computer Interface Design*, B Laurel [ed.], Addison Wesley, pp.209–219.

Norman, D A (1993), *Things That Make Us Smart*, Addison Wesley.

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

Polanyi, M (1966), *The Tacit Dimension*, Doubleday.

Pournelle (1990), *BYTE* September.

Repenning, A & Sumner, T (1992), "Using Agentsheets to Create a Voice Dialog Design Environment", in *Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing*, ACM Press, pp.1199–1207.

Rittel, H W J (1984), "Second-Generation Design Methods", in *Developments in Design Methodology*, N Cross [ed.], John Wiley & Sons Ltd, pp.317–327.

Schoen, D A (1983), *The Reflective Practitioner: How Professionals Think in Action*,, Basic Books.

Simon, H A (1981), *The Sciences of the Artificial*, MIT Press.

Stahl, G (1993), "Supporting Interpretation in Design", *Architecture and Planning Research, Special Issue on Computational Representations of Knowledge*, (in preparation).

Stefik, M J (1986), "The Next Knowledge Medium", *AI Magazine* 7 (1), pp.34–46.

Suchman, L (1987), *Plans and Situated Actions: The Problem of Human–Computer Communication*, Cambridge University Press.

Trigg, R H, Moran, T P & Halasz, F G (1987), "Adaptability and Tailorability in NoteCards", in *Proceedings of INTERACT'87 — Second IFIP Conference on Human–Computer Interaction*, H-J Bullinger & B Shackel [eds.], Elsevier Science (North Holland), pp.723–728.

Trigg, R H, Suchman, L A & Halasz, F G (1986), "Supporting Collaboration in NoteCards", in *Proceedings of CSCW'86: Conference on Computer Supported Cooperative Work*, D Peterson [ed.], ACM Press, pp.153–162.

Winograd, T & Flores, F (1986), *Understanding Computers and Cognition: A New Foundation for Design*, Ablex.