Formality Considered Harmful: Experiences, Emerging Themes, and Directions

Frank M. Shipman III and Catherine C. Marshall

CU-CS-648-93 April 1993

University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

FORMALITY CONSIDERED HARMFUL: EXPERIENCES, EMERGING THEMES, AND DIRECTIONS

Frank M. Shipman III

Department of Computer Science &
Institute for Cognitive Science
University of Colorado, Boulder, CO 80309-0430
(303) 492 - 1218
E-mail: shipman@cs.colorado.edu

Catherine C. Marshall

Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304
(415) 812 - 4740
E-mail: marshall@parc.xerox.com

ABSTRACT

This paper reviews experiences in designing, developing, and witnessing the shortcomings in a variety of systems. The authors anecdotally suggest that the cause of a number of unexpected difficulties in human-computer interaction is the user's unwillingness to make structure, content, or procedures explicit. Besides recounting experiences with system use, this paper discusses why users are often justified in rejecting formalisms and how system designers can anticipate and compensate for problems users have in making implicit aspects of their tasks explicit. Incremental and system-assisted formalization mechanisms, as well as techniques to evaluate the task situation, are proposed as approaches to this problem.

KEYWORDS: Formalization, structure, hypermedia, argumentation, design environments, knowledge-based systems, groupware, representation.

INTRODUCTION

Computer systems use abstract representations as the basis for user interaction for a variety of reasons - to structure a task or the user's work practices, to provide users with computational services like information management and retrieval functionality, or simply to make it convenient for the computer program to process the user's data. We refer to these abstractions as formalisms.

To work with formalisms embedded in computer systems, users engage in activities that might not ordinarily be part of their tasks - breaking information into chunks, characterizing information via keywords, categorizing information, or specifying how pieces of information are related to each other, for example. In terms of interacting with the Unix operating system, these activities might correspond to creating files, naming them, putting them in a directory structure, or making symbolic links between directories or files.

The abstract representations that computer systems impose on users may involve varying levels and types of formalization beyond what they are accustomed to. In some instances, little additional formalization is necessary to use a computer-based tool; text editors, for example, require minimal additional formalization beyond that demanded by other mechanisms for aiding in the production of linear text. Correspondingly, the computer can perform little additional processing without the use of heuristic techniques. In other cases, more formalization brings more computational power to bear on the task; idea processors and hypertext writing tools demand more specification of structure, but they also provide functionality that allows users to reorganize text or present it on-line as a non-linear work. We refer to these systems and their embedded representations as *semi-formal*, since they require some - but not complete - encoding of information into a schematic form. At the other end of the spectrum, formal systems require people to encode materials in a representation that can be fully interpreted by a computer program. When the level or type of formalization demanded by a computer system exceeds what the user expects, needs, or is willing to tolerate, the user will often reject the system.

In this paper, we suggest that creators of systems that support intellectual work like design, writing, or organizing and interpreting information are particularly at risk of expecting too great a level of formality from their users. To understand the effects of imposing or requiring formality, we draw on our own experiences designing and using such systems.

First, we give some anecdotal accounts of our own experiences as well as corroborative reports by others. We then discuss possible reasons why users reject formalisms, and propose some solutions for system designers who are trying to avoid making these same mistakes. In particular, we focus our proposals on mechanisms that are based on incremental system-assisted formalization and restructuring as people reconceptualize their tasks; we also consider ways designers can work with users to evaluate appropriate formalisms for the task at hand.

LEARNING FROM EXPERIENCE

The systems we discuss in this section have been successful by many metrics; yet they have all exhibited similar problems with user interaction that may be attributed to their underlying formalisms. To focus the discussion on these formalisms, we have deliberately left out any description of the interfaces by which users interact with the formalisms. In so doing, we hope to expose a dangerously seductive line of reasoning - that if you build the right interface to an embedded representation, users will formalize the desired aspect of their task domain.

What do systems supporting intellectual work require users to formalize? First, many hypertext systems try to coerce their users into making structure explicit. With few exceptions, they provide facilities for users to divide text or other media into chunks (usually referred to as nodes), and define the ways in which these chunks are interconnected (as links). This formalism is intended as either an aid for navigation, or as a mechanism for expressing how information is organized without placing any formal requirements on content.

Systems that support argumentation and the capture of design rationale go a step further than general-purpose hypertext systems in requiring users to formalize their information. They usually require the categorization of content within a prescriptive framework (for example, Rittel's Issue-Based Information Systems (IBIS) [16]) and the corresponding formalization of how these pieces of content are organized.

Other systems - in particular, knowledge-based systems - are built with the expectation of processing content. Thus, to add or change knowledge that the system processes, users are required to encode domain structure and content in a well-defined representational scheme. This level of formalization is built into a system with the argument that users will receive significant payback for this extra effort.

Groupware systems supporting coordination formalize something different from the structure of the information or its content. They expect a formalization of interactions between users of the system. This type of formalization allows the system to help coordinate activities between users, such as scheduling meetings or distributing information along a work-flow.

In this section, we examine each type of system - hypermedia systems that support the formalization of structure, knowledge-based systems that rely on the formalization of content, and groupware systems that require the formalization of procedure and interaction - with an eye toward how formalization influences system use and acceptance. We also look at activity-directed systems that have specific formalisms embedded in them, in this case, systems for the capture of argumentation and design rationale and design environments that combine semi-formal design rationale with more formal representations of domain knowledge.

Formalisms used in each of these types of systems involve computer-mediated communication or coordination with other humans, or the capture and organization of implicit knowledge. As a point of contrast, we also discuss results of efforts to support the software engineering process, systems designed specifically to aid in the production of a formalized artifact, a computer program.

General Purpose Hypermedia

Hypermedia systems generally provide a semi-formal representation where chunks of text or other media can be connected via navigational links. The goal of these links is to better provide for individualized reading patterns through the non-linear traversal of the document. Authors formalize structure during the creation of such hyperdocuments.

Learning how to write, and to a lesser extent learning how to read, in a hypertext system takes time. Although the first author had used NoteCards [13] previously, his first real experience with writing in a hypertext system was with KMS [1]. He spent several months writing hierarchical outlines and full pages of text connected by a single link to the next page of text, as if he was still using an outlining tool and word processor. By defaulting to his previous experience, he did not have to decide what information should be chunked together or what links should be created. Information that fit on a page became a chunk with a link to the next page.

Many NoteCards users reported similar problems. The second author's experiences training information analysts to use NoteCards showed that they had difficulties chunking information into cards ("How big is an idea? Can I put more than one paragraph on a card?"), naming cards ("What do I call this?"), and filing cards ("Where do I put this?"). Typed links - the strongest formalization mechanism provided - were rarely used, and when they were, they were seldom used consistently. Both link direction and link semantics proved to be problematic. For example, links nominalized as "explanations" sometimes connected explanatory text with the cards being explained; other times, the direction was reversed. Furthermore, the addition of "example" links confounded the semantics of earlier explanation links; an example could easily be thought of as an explanation. Monty documents similar problems in her observations of a single analyst structuring information in NoteCards in [24]

Aquanet [21] has a substantially more complex model of hypertext that involves a user-defined frame-like knowledge representation scheme with a graphical presentation component. We observed that even sophisticated users with a background in knowledge representation had problems formalizing previously implicit structures. A case study of a large-scale analysis task documents these experiences in [22].

Argumentation and Design Rationale

Recently there have been many different proposals for embedding specific representations in systems to capture argumentation and design rationale. Some of them use variations on Toulmin's micro-argument structure [29] or Rittel's issue-based information system (IBIS) [16]; others invent new schemes like Lee's design representation language [17] or MacLean et al's Question-Option-Criteria [18].

The benefits of having a formal argumentation or design rationale method have been almost grail-like in scope: shorter production time, lower maintenance costs on products, and better designs are just a few of the promises [14]. There have been a number of applications of these mechanisms, from McCall et al's use of PHI [23] to Yakemovik and Conklin's use of itIBIS [31]. The results can be interpreted both as successes and as failures. On one hand, by some measures, the methods did result in long-term costs reductions. On the other hand, only severe social pressure, extensive training, or continuing human facilitation seems to be adequate to get most people to use the methods. In fact, in [6], Conklin and Yakemovik report that they had little success in persuading other groups to use itIBIS outside of Yakemovik's development team, and that meeting minutes had to be converted to a more conventional prose form to engage any of these outside groups.

Like the general-purpose hypermedia systems, these systems force their users to divide information into chunks which are categorized as certain types, such as *issue*, *position*, or *argument*. Users of these methods must then specify connections between chunks, such as *answers*, *supports*, or *contradicts* links. We have noticed several problems users have in effectively formalizing their design rationale or argumentation in this type of system; these problems can be predicted from our experiences with hypertext.

First, people aren't always able to chunk intertwined ideas; we have observed, for example, positions with arguments embedded in them. Second, people seldom agree on how information can be classified and related in this general scheme; what one person thinks is an argument may be an issue to someone else. Both authors have engaged in extended arguments with their collaborators on how pieces of design rationale or arguments were interrelated, and about the general heuristics for encoding statements in the world as pieces of one of these representation schemes (see [21] for a short discussion of collaborative experiences using Toulmin structures). Finally, there is always information that falls between the cracks, no matter how well thought out the formal representation is. Conklin and Begeman document this as well in their experiences with gIBIS [5].

Knowledge-Based Design Environments

Design environments consist of a number of components integrated to support the process of design [7]. These design environments include different mechanisms for representing knowledge, including formally represented domain knowledge, semi-formal argumentation and informal textual annotations.

This variety of knowledge representations has led to the development of different mechanisms for supporting the modification of knowledge in the systems. One such mechanism is the set of end-user modifiability (EUM) tools developed to support designers in modifying and creating formal domain knowledge with task agendas, explanations, and examples [8]. In a description of user studies on EUM tools Girgensohn says that most of the problems found in the last round of testing "were related to system concepts such as classes or rules" [10]. In short, one finding of these user studies was that, although the EUM tools made the task significantly easier, people still had problems in dealing with the formalisms imposed by the underlying system.

Knowledge-Based Systems

Knowledge-based systems have long exclaimed the goal of having users add or correct knowledge in the system. End-user knowledge acquisition imposes formalization requirements on users that are similar to those imposed by design environments except that they lack much of the support provided by the EUM tools. Users must learn the knowledge representation used by the system, even if it is hidden by a good interface, so they may understand the effects of their changes.

A different approach to the problem of creating user modifiable expert systems was taken by Peper et al. in [26]. They removed the inference engine and just created a hyperdocument where the user was asked a question and based on their answer, they went to a new point in the document. This meant that users could add new questions or edit old questions by using English since the computer was not doing any processing over the information. By reducing the need for formalized knowledge, they realized an advantage in producing a modifiable system.

Groupware Systems

Groupware systems that require the formalization of procedure and interaction have suffered many of the same problems as systems that enforce formalization of structure and content. For example, systems that extend electronic mail by attaching properties or types to messages require their users to classify exactly what type of message they are sending or what type of reply is acceptable. Experiences with systems like the Coordinator [30] and Information Lens [20] point out that many users ignore the formal aspects of such systems, and generally use them as basic electronic mail systems [4].

Other groupware systems also exhibit this property. Systems that require keywords selected from a limited vocabulary, even one as rich as the Medical Subject Headings (MeSH) used by medical journals, also force users to express concepts in a language which may lack the terms that they had previously applied to their work.

Coordination oriented systems have the additional burden of formalizing social practices which are largely left implicit in normal human-human interactions. Automatic scheduling systems have met with limited acceptance [12] due to the unwillingness of users to describe their normal decision methods for whether and when to schedule a meeting with other people. The same rules of scheduling that apply to your boss do not apply to an unknown person, but formalizing such differences is difficult.

Software Engineering

Software engineering echoes the difficulties described above. In both cases people are required to explicitly communicate information to a computer. The interfaces through which this communication occurs are often part of the problem, but they only contribute what Brooks calls "accidental complexity" [3] to the overall task. Whether a person uses popup menus, dialog boxes, "English-like" formal languages, or low level programming languages to state the information explicitly, the person must still know what they want to state, be it a relationship between two pieces of text or a complex algorithm.

In software engineering, deciding what needs to be stated explicitly (the specification) has been termed "up-stream activity". The software technology program at MCC was explicitly charged by its director, Les Belady, to create support for tasks leading up to specification [2]. The resulting work supports the process of coming up with a specification, the storage and retrieval of information associated with this process, and visualization of the result. These same goals could be used to focus work on supporting formalization in other domains.

WHY USERS SHOULD NOT FORMALIZE

From the above discussion we hope to impart the notion of how endemic we believe the problems of expecting formalization are. In this section we explain why we believe that the users are making the right decisions, in some sense, by resisting premature, unnecessary, meaningless, or cognitively expensive formalization.

From the user's perspective formalization poses many risks. "What if I commit to this formalization only to later find out it is wrong?" "What do I do when the ideas or knowledge is tacit and I cannot formalize it?" "Why should I spend my time formalizing this when I have other things to be doing?" "Why should I formalize this when I cannot agree with anyone else on what the formalization should be?" These are all valid questions and the answers that systems provide are often insufficient to convince people to use a system's formal aspects.

Imposing Unnecessary or Premature Structure

One well known reason why users will not formalize is the negative effects of prematurely or unnecessarily imposing a structure. This cause was noted by Malone in his studies of how people organized information in their offices [19]. One problem he found is that in many cases trying to create a new formalization from information in a previous formalization is more difficult than formalizing the information from an informal state. Malone's study found some people having piles of papers waiting to be filed because the person did not yet know where to file them. In essence, the difficulty associated with undoing the formalization - in this case, filing - may compare with the formalization process itself.

Another problem associated with unnecessary structure is the perception that information that is formalized incorrectly or inconsistently will be of less use (or at least more expensive to use) than information not formalized. This can be seen in the directory structures of UNIX, Mac OS, or DOS users who have huge numbers of files at the top level directory (or filebox) of their machine or account. Their opinion is "Sure, if I organized these files into the right hierarchy I might have less difficulty in dealing with them, but how would I ever find anything that I put in the 'wrong' place?" Functionality such as the notorious UNIX 'find' command has been created in an attempt to help users with such problems, but many users find learning about the intricacies of such commands or tools unacceptable.

Analytic Structure vs. Generative Structure

Many of the representations that designers have embedded in systems are the result of an analysis of existing material. For example, argument representations are often derived from analyzing existing argumentative discourse, and classifying discourse units in categories, then describing in general terms how these categories are related. But post hoc analysis is very different from generation. When these descriptive models are given to users, they find it very difficult to formalize knowledge as they are generating or producing it. An example is the observation in [9][8] that design students have difficulty producing IBIS-style argumentation even though videotapes of their design sessions show that their naturally occurring discussions follow this structure.

This relates to the more general difficulty in expressing tacit knowledge. The introspection necessary to produce and apply a formal representation during a task necessarily interrupts the task, structures and changes it. As a simple example, if a person is asked to describe what it means to breath normally, they will probably be unable to continue to do so. Furthermore, chances are that introspection about what normal breathing means will cause the person's breathing to become abnormal - exaggeratedly shallow, overly deep, irregular.

This simple example predicts the second author's experience with Aquanet. In Aquanet, users are faced with a meta-task of describing how their domain is structured or choosing among an existing library of schematic structures. Experience suggests that users can not (or will not) articulate how their tasks are formalized at the outset. Once a formalization is selected, users have a tendency to use as few distinctions and as little structure as possible, but we still feel that the formality of the representation shapes both the kind of information collected and how it is interpreted [22].

Cognitive Inflation

Adding information to a system is cognitively expensive; adding formal knowledge breaks the bank. Just trying to figure out how a domain is organized to collect and categorize semi-structured notes or classify email messages is difficult enough, but formalizing that information so that a computer can reason about it is much worse. First of all the user must learn the computer's language. Some limited domains, such as circuit design, have formal languages to describe a certain type of information. General purpose formal languages, such as languages for frames and semantic nets, are almost never used for tasks not dealing with a computer.

Reasons why these formalisms are too difficult for people to use often concern the many extra decisions that they require to specify anything. Many of these extra decisions concern chunking, linking, and labeling. People spend years learning how to divide up ideas into sentences and paragraphs in natural language, and formal languages require much more explicitly defined boundaries, connections between chunks, and labels for such connections.

In an experiment in applying Assumption-based Truth Maintenance Systems (ATMS) derived dependency analysis (described in [15]) to networks of Toulmin micro-argument structures in NoteCards, one of the authors came to the conclusion that the cognitive cost was not commensurate with the results, even though dependency analysis had long been a goal of explicitly representing the reasoning in arguments. Although the hypertext representation of the informal syllogistic reasoning inherent to Toulmin structures (the data-claim-warrant triple) captures a dependency relationship, additional formalization is necessary to perform automated analysis by an ATMS model. In particular, it was important to identify assumptions, and contradictory nodes. Not only was it difficult to identify contradictions in real data (belief was qualified rather than absolute) and impossible to track relative truth values over time, but also - and most importantly - by the time contradictions had been specified and relative truth values had been determined, a signification portion of the network evaluation had been done by the user. In this case, the additional processing done by the ATMS mechanism added little value.

Different People, Different Tasks, Different Structure

The difficulties of creating useful formalizations for individual use are compounded when different people must share the formalization. An analogy can be drawn between collaborative formalization and writing a legal document for multiple parties who have different goals. The best one can hope for in either case is a result sufficiently vague that it can be interpreted in an acceptable way to all the participants; ambiguity and imprecision are used in a productive way. Formalization makes such agreements difficult because it requires the formalized information to be stated explicitly so that there is little room for different interpretations.

For different people to agree on a formalization they must agree on the chunking, the labelling, and the linking of the information. As we have shown by earlier examples in the use of tools to capture design rationale, the prospects of negotiating how information is encoded in a fixed representation is at best difficult.

This problem does not occur just when multiple people use the same structure but can also occur when the creator of the structure has a different task than before. The context of the new task may not match well with the structuring scheme.

Anecdotal evidence shows that a representation that is suitable for one task may not be appropriate for a very similar related task. For example, in [22] we describe how a representation developed for an analytic task - an assessment of foreign machine translation efforts - proved to be of limited value in a very closely related task, evaluating Spanish-English machine translation software. The second task shared a subset of the content with the first task, but the representation did not formalize appropriate aspects of the material. Attributes like speed and accuracy as well as cost and computer platform turned out to be very important in evaluating software, but only of secondary importance in a general assessment of the field, while in the general assessment of the field, the technical approach of the various systems was deemed important. In short different structures will be of use in different situations [28].

LEARNING TO ANTICIPATE USER NEEDS

While difficulties caused by formalization are widespread and users are justified in their resistance to or rejection of some formalization tasks, there are some partial solutions to this dilemma for system designers. First, designers should decide what information must be formalized for the task to be performed and provide for that. Second, designers should decide what other services or user benefits the computer can provide based on trade-offs introduced by additional formalization. Finally, designers should expect, allow, and support reconceptualization and incremental formalization in longer tasks.

Essentials for Task

Some information must be formalized for the computer system to perform almost any task. A word processor must know the order of characters, a drawing program must know the color and shape of objects being drawn, and a circuit analyzer must know the logical circuit design. Interaction based on a limited-domain formalism can become transparent when the user has become skilled in the formalism. Failure to get the user to formalize information that is essential for the central task means rejection of the system.

But what is the central task for more general-purpose systems and, informationally, what does it require? What must by formalized for a system to support the organization and sharing of information? Does the content just have to be entered into the system, or for the system to work does extra information, such as hypertext links and labels need to be specified? To answer these questions, participatory design techniques can be applied to gain an understanding of the users' work practices and the formalisms necessary to support these practices [11].

The Non-Essential Cost/Benefit Trade-off

Many systems provide features which are not necessary for some uses of the system but are available for users who want the added benefits of providing more information. Font style and size could be considered such information in a word processor. Users can accept the default style and size to write a paper, and thus never have to explicitly state their preference, but there is the option for different fonts. Certainly many people seemed very happy to take advantage of this particular feature, placing many fonts on every page until some notion of aesthetics became common.

Other features may be much less widely used. Spreadsheet programs include many features which are used only by a small percentage of the user community [25]. The rest of the users either get by without using the features, or ask for help when they cannot avoid doing otherwise. In designing information systems where formalization is required for use of some of the features, systems designers must balance the effects of cognitive inflation, which can leave your services worth little compared to the cost of formalization.

One system design goal can be to provide functionality based on inferred structures in informally represented information; structures can be inferred by spatial, textual, temporal, or other patterns. The system's inferences will be incorrect at times but, as long as the inferences are right part of the time and it is apparent to the user when the system has made the wrong inference, these features will cost the user little for the benefit they provide.

Gradual Formalization and Restructuring

Two of the reasons why users resist formalization, namely the threat of premature formalization and inability to formalize with current understanding, will change over time. Furthermore, tasks are frequently reconceptualized during performance. This is why systems where incremental formalization and restructuring can occur seem to be necessary.

Supporting gradual formalization can be more than just allowing users to add formalized information when the urge hits them. Systems can be designed to support the process of formalization, particularly the process of collaborative formalization. Systems can also make suggestions about what formalizations might be appropriate by noticing patterns in informally represented information [27]. As with providing services based on inferred structure, suggestions based on such inferences do not have to be correct all the time; the user just needs to be able to know when to accept them and when not to. Suggestions can also lower the cost of defining structure, by providing an initial formalization which can then be modified rather than having to be created from scratch.

CONCLUSIONS

We have tried to describe the extent of the difficulties caused by systems that require users to formalize information. These problems are pervasive in systems designed to support intellectual work such as hypermedia, argumentation, knowledge-based systems, and groupware.

The difficulties that users have in formalizing information is not just an interface problem. Users are hesitant about formalization because of a fear of prematurely committing to a specific perspective on their tasks; it may also be difficult for them to formalize knowledge that is usually tacit. The added cost of formalizing information over using informal information makes formalized information far less attractive for users to provide. In a collaborative setting, people must agree on a formalization and the heuristics for encoding information into it.

There are decisions that system designers can make to reduce the need of formal information by systems and also methods to reduce the difficulty for users in providing this information. Systems should use the domain-oriented representations used to communicate unambiguously between humans when possible. Systems should provide services based on inferred structure in informally represented information. Finally, systems should support the process of incremental formalization and structure evolution as tasks are reconceptualized.

As system designers, it is tempting to add more whiz-bang features which rely on formalized information. We must temper that urge and consider the difficulty that the user will have providing that information before counting on it for the success of our systems.

ACKNOWLEDGEMENTS

We thank the members of the HCC group at the University of Colorado and the Collaborative Systems Area at Xerox PARC for discussions that have aided in the formation of these ideas. We also thank Jonathan Grudin and Tom Moran for reading and providing comments on early versions of this paper. This research was supported in part by grant No, IRI-9015441 from the National Science Foundation.

REFERENCES

1. Akscyn, R.M., McCracken, D.L., Yoder, E.A. "KMS: A Distributed Hypermedia System for Managing Knowledge in Organizations". *Communications of the ACM* 31, 7 (July 1988), 820-835.
2. Belady, L. "MCC: Planning the Revolution in Software". *IEEE Software* (November 1985), 68-73.
3. Brooks Jr., F.P. "No Silver Bullet: Essence and Accidents of Software Engineering". *IEEE Computer* 20, 4 (April 1987), 10-19.
4. Bullen, C.V., Bennett, J.L. Learning From User Experience With Groupware. *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW'90)*, New York, 1990, pp. 291-302.
5. Conklin, E.J., Begeman, M.L. gIBIS: A Hypertext Tool for Exploratory Policy Discussion. MCC Technical Report Number STP-082-88, Austin, Texas, 1988.
6. Conklin, E.J., Yakemovic, K.C. "A Process-Oriented Approach to Design Rationale." *Human Computer Interaction* Vol. 6, No. 3 & 4, 1991, pp. 357-391.
7. Fischer, G., Grudin, J., Lemke, A.C., McCall, R., Ostwald, J., Reeves, B.N., Shipman, F. "Supporting Indirect, Collaborative Design with Integrated Knowledge-Based Design Environments". *Human Computer Interaction* 7, 3 (1992). (in press).

8. Fischer, G., Girgensohn, A. End-User Modifiability in Design Environments. Human Factors in Computing Systems, *CHI'90 Conference Proceedings* (Seattle, WA), ACM, New York, April, 1990, pp. 183-191.

9. Fischer, G., Lemke, A.C., McCall, R., Morch, A. Making Argumentation Serve Design. *Human Computer Interaction*, 6, 3-4, 1991, pp. 393-419.

10. Girgensohn, A. End-User Modifiability in Knowledge-Based Design Environments. Ph.D. Dissertation., Department of Computer Science, University of Colorado, Boulder, CO, 1992.

11. Greenbaum, J., Kyng, M. (Eds.) *Design at Work: Cooperative Design of Computer Systems*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1991.

12. Grudin, J. Why CSCW Applications Fail: Problems in the Design and Evaluation of Organizational Interfaces. *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW'88)*, ACM, New York, September, 1988, pp. 85-93.

13. Halasz, F.G., Moran, T.P., Trigg, R.H. NoteCards in a Nutshell. Human Factors in Computing Systems and Graphics Interface, *CHI+GI'87 Conference Proceedings* (Toronto, Canada), ACM, New York, April, 1987, pp. 45-52.

14. Jarczyk, A,. Loeffler, P., Shipman, F. Design Rationale for Software Engineering: A Survey. *Proceedings of the 25th Hawaii International Conference on System Sciences*, 1992, pp. 577-586.

15. de Kleer, J. "An Assumption-based TMS" *Artificial Intelligence* 28, 1986, pp. 127-162.

16. Kunz, W., Rittel, H.W.J. Issues as Elements of Information Systems. Working Paper 131, Center for Planning and Development Research, University of California, Berkeley, CA, 1970.

17. Lee, J. SIBYL: A Tool for Managing Group Decision Rationale. *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW '90)*, New York, October, 1990, pp. 79-92.

18. MacLean, A., Young, R., Bellotti, V., Moran, T. "Questions, Options, and Criteria: Elements of a Design Rationale for User Interfaces". *Human Computer Interaction* Vol. 6, No. 3 & 4, 1991, pp. 201-250.

19. Malone, T.W. "How do People Organize their Desks? Implications for the Design of Office Information Systems". *ACM Transactions on Office Information Systems* 1, 1 (January 1983), 99-112.

20. Malone, T.W., Grant, K.R., Lai, K.-Y., Rao, R., Rosenblitt, D. Semi-Structured Messages are Surprisingly Useful for Computer-Supported Coordination. *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW'86)*, MCC, Austin, TX, December, 1986, pp. 102-114.

21. Marshall, C., Halasz, F., Rogers, R., Janssen, W. Aquanet: a hypertext tool to hold your knowledge in place. Hypertext '91 Conference, 1991, pp. 261-275.

22. Marshall, C.C., Rogers, R.A. Two Years before the Mist: Experiences with Aquanet. To appear in *Proceedings of European Conference on Hypertext (ECHT '92)*, Milano, Italy, December 1992.

23. McCall, R., Schaab, B., Schuler, W. An Information Station for the Problem Solver: System Concepts. *Applications of Mini- and Microcomputers in Information*, Keren, C., Perlmutter, L. (eds.), Elsevier, New York, 1983.

24. Monty, M.L. Issues for Supporting Notetaking and Note Using in the Computer Environment. Dissertation, Department of Psychology, University of California, San Diego, 1990.

25. Nardi, B.A., Miller, J.R. An Ethnographic Study of Distributed Problem Solving in Spreadshet Development. *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW'90)*, 1990, pp. 197- 208.

26. Peper, G., MacIntyre, C., Keenan, J. Hypertext: A New Approach for Implementing an Expert System. *Proceedings of 1989 ITL Expert Systems Conference*, 1989.

27. Shipman, F. Supporting Knowledge-Base Evolution using Multiple Degrees of Formality. Tech. Rept. CU-CS-592-92, Department of Computer Science, University of Colorado, Boulder, CO, 1992.

28. Suchman, L.A. *Plans and Situated Actions*. Cambridge University Press, Cambridge, UK, 1987.

29. Toulmin, S. (Ed.) *The Uses of Argument*. Cambridge University Press, UK, 1958.

30. Winograd, T., Flores, F. *Understanding Computers and Cognition: A New Foundation for Design*. Ablex Publishing Corporation, Norwood, NJ, 1986.

31. Yakemovic, K.C., Conklin, E.J. Report of a Development Project Use of an Issue-Based Information System. *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW'90)*, 1990, pp. 105-118.