
Computer Supported Cooperative Work **8:** 333–352, 1999.

Formality Considered Harmful: Experiences, Emerging Themes, and Directions on the Use of Formal Representations in Interactive Systems

FRANK M. SHIPMAN III & CATHERINE C. MARSHALL
Department of Computer Science, Texas A&M University, College Station, TX 77843-3112

(Received August 13 1997)

Abstract. This paper reflects on experiences designing, developing, and working with users of a variety of interactive computer systems. The authors propose, based on these experiences, that the cause of a number of unexpected difficulties in human-computer interaction lies in users' unwillingness or inability to make structure, content, or procedures explicit. Besides recounting experiences with system use, this paper discusses why users reject or circumvent formalisms which require such explicit expression, and suggests how system designers can anticipate and compensate for problems users have in making implicit aspects of their tasks explicit. The authors propose computational approaches that address this problem, including incremental and system-assisted formalization mechanisms and methods for recognizing and using undeclared structure; they also propose non-computational solutions that involve designers and users reaching a shared understanding of the task situation and the methods that motivate the formalisms. This paper poses that, while it is impossible to remove all formalisms from computing systems, system designers need to match the level of formal expression entailed with the goals and situation of the users – a design criteria not commonly mentioned in current interface design.

Key words: formalization, structure, hypermedia, argumentation, design environments, knowledge-based systems, groupware, knowledge representation, tacit knowledge

1. Introduction

Systems that support collaborative work provide informational and social structures through which communication and coordination occur. The creation of these structures – especially the tendency to require explicit statements of these structures – has been an issue for groupware ranging from tools like group calendars (Grudin, 1988) and workflow systems (Ellis, Gibbs and Rein, 1991) to systems addressing general practices of categorization and social coordination (Suchman, 1994; Winograd, 1994; Bannon, 1995). Indeed, Schmidt and Bannon describe the ability to articulate information used in collaborative work and information about the work itself, as central to the success of CSCW systems (Schmidt and Bannon, 1992). By reflecting on experiences with a variety of systems, we argue that the

use of formal representations hinders this articulation thus causing many of the difficulties encountered by CSCW systems.

When people use computer systems, their interaction is usually mediated by abstract representations that describe and constrain some aspect of their work or its content. Computer systems use these abstract representations to support their users' activities in a variety of ways: by structuring a task or users' work practices, by providing users with computational services such as information management and retrieval, or by simply making it possible for the system to process users' data. These abstractions are frequently referred to as formalisms.

When formalisms are embedded in computer systems, users often must engage in activities that might not ordinarily be part of their tasks: breaking information into chunks, characterizing content with a name or keywords, categorizing information, or specifying how pieces of information are related. For example, in the World Wide Web, these activities might correspond to creating pages, giving them titles or other metadata, putting the pages into a hierarchical directory structure, and adding navigational links between the pages.

The abstract representations that computer systems impose on users may involve varying degrees and types of formalization beyond those that users are accustomed to. In some instances, little addition formalization is necessary to use a computer-based tool; text editors, such as vi or emacs, do not require additional formalization much beyond that demanded by other mechanisms for aiding in the production of linear text. Correspondingly, the computer can perform little additional processing without applying sophisticated content analysis techniques. In other cases, more formalization brings more computational power to bear on the task; idea processors and hypermedia 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. These systems and their embedded representations are referred to as semi-formal since they require some – but not complete – encoding of information into a schematic form. At the formal end of the spectrum, knowledge-based systems require people to encode materials in a representation that can be fully interpreted by a computer program.

In this paper, we describe how 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 draw lessons from some of our experiences with these types of systems as well as corroborative reports by others. We discuss possible reasons why users reject formalisms, including issues associated with cognitive overhead, tacit knowledge, premature structure, and situational structure. We then propose system design approaches that address the problems associated with formalisms. 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.

2. Experiences with a variety of formalisms

To understand how formalization influences system use and acceptance, this paper examines four different kinds of systems that support intellectual work: general purpose hypermedia systems, systems for capturing argumentation and design rationale, knowledge-based systems, and groupware. Some of these systems, such as those designed to capture design rationale, are based on specific formalisms that reflect a prescriptive method or approach to the work; others, such as hypermedia systems, require that an arbitrary formal structure be developed given more abstract, less prescriptive, building blocks. Each type of system addresses a very different aspect of a user's work, but all advance our analysis of the underlying problems developers encounter when they field systems to support intellectual work.

What do systems supporting intellectual work require their users to formalize? First, many hypermedia models, including the World Wide Web, are designed to allow authors to make structure explicit. They provide facilities for authors to divide text or other media into chunks (usually referred to as nodes or pages), 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: they usually require the categorization of a content within a prescriptive framework (for example, in Rittel's Issue-Based Information Systems (IBIS) (Kunz and Rittel, 1970) and subsequent derivatives, users must specify whether a given chunk is an issue, a position, or an argument) and the corresponding formalization of how these pieces of content are organized (for example, which argument supports which position). This prescriptive framework is seen as providing a facilitative methodology for the task.

Knowledge-based systems are built with the expectation of processing content (Waterman, 1986). Thus, to add or change knowledge that the system processes, users are required to encode domain structure and content in a knowledge representation language. This level of formalization enables the system to apply knowledge-based reasoning techniques to support users by performing tasks such as automated diagnosis, configuration, or planning.

Groupware systems supporting coordination are an interesting counterpoint to knowledge-based systems: they may not require users to formalize the structure of their information or its content, but rather their own interactions. This type of formalization allows the system to help coordinate activities between users, such

as scheduling meetings or distributing documents along a workflow (Ellis, Gibbs and Rein, 1991).

Through analysis of this range of information systems, we describe how formalisms structure the activity in unexpected or unintended ways, are understood differently by different people, may be an unfamiliar addition to a formerly familiar task, may cause people to lose information that falls outside the prescribed structure, and in general require people to make knowledge explicit that may be difficult or undesirable to articulate. Our discussions delve into these lessons, and illustrate them with specific experiences.

2.1. General purpose hypermedia

Hypermedia systems, including the World Wide Web, provide a semi-formal representation where chunks of text or other media, called nodes, can be connected via navigational links. An important goal of these links is to accommodate individual reading patterns by supporting non-linear traversal of the document. Authors must formalize structure during the creation of such hyperdocuments.

Learning how to write, and to a lesser extent learning how to read, in a hypermedia system takes time. Observing writers become accustomed to page-based hypertext (WWW, NoteCards, KMS, and VNS), it became apparent that people do not easily accept new authoring modes. In these page-based hypermedia systems, authors record information on electronic pages which can be linked together with navigational links. Frequently, novice authors begin by following practices from prior authoring tools, such as outlining tools or word processors. Thus, hypertexts end up as hierarchical outlines with full pages of text connected by a single link to the next page of text. By defaulting to the authoring practice of familiar systems, users avoid the decision of what information belongs together in a node or what links should be created. Information that fit on a page became a chunk with a link to the next page. Thus the new medium, with its unfamiliar formalism, combined with existing practice to yield unexpected results.

NoteCards (Halasz, Moran and Trigg, 1987) is a hypermedia environment that uses an index card metaphor. Authoring in NoteCards involves deciding how much content to put in each card, naming individual cards, and filing cards into electronic fileboxes; the system supports and enforces this working style. Not surprisingly, many NoteCards users reported problems creating non-linear structures in the unfamiliar medium. Experiences training information analysts to use NoteCards revealed 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? Do I have to name this card before I can get it off the screen?”), and filing cards (“Where do I put this?”). Typed links – the strongest formalization mechanism the system 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 (Monty, 1990). She describes her observations of a subject taking notes in preparation for writing a paper:

> The processes of creating a note, titling it, filing it in a FileBox, and creating a link . . . were sometimes difficult for the subject. Many times he struggled to create a title for his note; he often claimed that the most difficult aspect of this task was thinking of good titles (Monty, 1990, p. 71).

She confirms our own observations of information analysts and their use of links, "In his earlier notetaking using NoteCards, he [the subject] was more likely to link notes together. As time went on though he built fewer special purpose links between cards and relied on Source links and filing in FileBoxes [the primary system-supported link types]." Of course, training and supervision helped users learning the general techniques for hypermedia authoring, but they tended to avoid (or lose interest in) the more sophisticated formalisms.

Unlike NoteCards, which only supported the expression of binary relationships between chunks of information, Aquanet (Marshall et al., 1991) used a substantially more general (and more complex) model of hypertext that involved a user-defined frame-like knowledge representation scheme with a graphical component. Aquanet users would first select a schema, a description of the node and relation (link) types to be used in a particular information space, and then begin creating instances of these types of a two-dimensional plane. Node types generally had distinctive visual properties; relations could have visible manifestations and impose layout constraints on the nodes they connected.

Experiences with use showed that instead of building large interconnected networks of nodes (like the designers expected), users created linkless spaces of nodes arranged in regular graphical patterns that indicated relationships among nodes spatially and visually (although the structures were well within the range of Aquanet's more formal object/relation model) (Marshall and Rogers, 1992). In fact, Aquanet's greatest strength ended up being its ability to express interpretations in terms of visual appearance and spatial positioning; users chose informal modes of expression that circumvented the more powerful knowledge representation mechanism.

Experiences with internal use of early prototypes of the Virtual Notebook System (VNS) (Shipman, Chaney and Gorry, 1989), another page-based hypermedia system, point out the potential for work practices to emerge which reduce the difficulties of chunking and linking information. Groups agreed upon high-level organizational conventions so they could locate and understand information in each other's notebooks. These high-level conventions did not inhibit individual variations in the amount of information on a page and the number of links creates.

Later usage of the VNS outside of the development community showed how user communities adapted to the new technology (Brunet, Morrissey and Gorry 1991). Over time, users built up structured templates similar to form-based interfaces, thus reducing the overhead involved in adding structure to the information they were entering. In this case, the new medium placed additional requirements on the task; the use of formalisms had to be negotiated within the workgroup. Users, rather than developers, designed their own formalisms to match their task.

2.2. Argumentation and design rationale

There have been many different proposals for embedding specific representations in systems to capture argumentation and design rationale (HCI Journal, 1991). Some of them use variations on Toulmin's micro-argument structure (Toulmin, 1958) or Rittel's issue-based information system (IBIS) (Kunz and Rittel, 1970); others invent new schemes like Lee's design representation language (Lee, 1990) or MacLean and colleagues' Question-Option-Criteria (MacLean et al., 1991).

The goals of formal argumentation or design rationale include lower maintenance costs on products, and better designs due to the earlier discovery of inconsistencies and miscommunications. Reported experiences with mechanisms to capture design rationale – from McCall et al.'s use of PHI (McCall, Schaab and Schuler, 1983) to Conklin and Burgess Yakemovic's use of itIBIS (Conklin, and Burgess Yakemovic, 1991) – can be interpreted as limited successes. The methods resulted in long-term cost reductions, but success relied on social pressure, extensive training, or continuing human facilitation. In fact, Conklin and Burgess Yakemovic reported that they had little success in persuading other groups to use itIBIS outside of Burgess Yakemovic'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 general-purpose hypermedia systems, argumentation and design rationale systems prescribe that their users chunk and categorize information according to its methodological role, such as issue, position, or argument. Users of these methods must then specify connections between chunks, such as answers, supports, or contradicts links. Both authors have, independently, worked with argumentation schemes and 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 the prior experiences with hypermedia.

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 (Newman and Marshall, 1992) 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 the latter problem in their experiences with gIBIS (Conklin and Begeman, 1988).

2.3. KNOWLEDGE-BASED SYSTEMS

Knowledge-based systems have long endorsed the goal of having users add or correct knowledge in the system. End-user knowledge acquisition imposes a number of formalization requirements on users. Users must learn the system's knowledge representation, even if it is hidden by a good interface, or else they will not fully understand the effects of their changes.

One approach to enabling end-user knowledge acquisition is to have knowledge engineers create domain-oriented knowledge acquisition tools. Domain-oriented tools are designed to allow users to specify information in familiar abstractions. Because creating such domain-oriented tools is time-consuming, domain-independent meta-level tools, such as PROTEGE (Musen, 1989) and DOTS (Eriksson, 1991), have been developed to support the creation of the domain-oriented knowledge acquisition tools. Even after such domain-oriented support is available, users are left to work through the interdependence of rules and to decide how to abstract from specific instances to more useful general information.

A second approach to supporting end-user knowledge acquisition is demonstrated by the end-user modifiability (EUM) tools developed to support designers in modifying and creating formal domain knowledge. Tools such as task agendas, critics, explanations, and examples allowed end-users to more effectively modify a knowledge-base without the intervention of a knowledge engineer (Girgensohn, 1992). In a description of user studies on EUM tools, Girgensohn notes that most of the problems found in the last round of testing "were related to system concepts such as classes or rules." In short, these user studies revealed that, although the EUM tools made the input of knowledge significantly easier, users still had problems manipulating the formalisms imposed by the underlying system.

Peper and colleagues developed a third approach to the problem of creating expert systems that users can modify (Peper et al., 1989). They eliminated the inference engine, leaving a hypermedia interface in which users were asked questions and based on their answers, were directed to a new point in the document. For example, a user might see a page asking the question. "Did the warning light come on?" with two answers, "Yes" and "No". Each answer is a link to further questions or information based upon the answer of the previous question. With this system, users could add new questions or edit old questions in English since the computer was not processing the information. By reducing the need for formalized knowledge, they produced a modifiable system although they sacrificed inferencing.

2.4. 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 of 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 (Winograd and Flores, 1986) and Information Lens (Malone et al., 1986) point out that many users ignore the formal aspects of such systems, and generally use them as basic electronic mail systems (Bullen and Bennett, 1990).

Coordination oriented systems have the additional burden of formalizing social practices which are largely left implicit in normal human-human interactions. Automatic scheduling systems, for example, have met with limited acceptance (Grudin, 1988); users have proven to be unwilling to describe how they decide whether and when to schedule meetings with other people. Scheduling rules that apply to one's manager do not apply to a stranger; making such differences explicit is not only difficult, but also socially undesirable.

Experiences with workflow systems, systems which automatically route documents and work through defined procedures, show that systems without the ability to handle exceptions to the formalized procedure cannot support the large number of cases when exceptional procedures are required (Ellis, Gibbs and Rein, 1991). Arguably, almost all office procedures turn out to be exceptions to the prescribed form (Suchman, 1987). Increasing the flexibility in representing group processes has been the goal behind much of the recent research in workflow systems (Glance, Pagani and Pareschi, 1996; Dourish et al., 1996).

Furthermore, choosing which procedures to encode can be difficult. Do the procedures written in the corporate manual get encoded, or those that are actually followed? How are the actual procedures obtained? Does the encoding of the actual procedures given them legitimacy that will be resisted by those who define or follow the corporate procedures? Formalization of such information can quickly lead to a political battle whose first casualty is the workflow system. In this light it is not surprising that the current enthusiasm with workflow systems centers around business-process reorganization (BPR) – the replacement of existing and practiced procedures of operation.

3. Difficulties arising from formal interactions

The broad range of examples discussed in the previous section highlights the ubiquity of the problems associated with enforced formalization. This section explains some of the problems that frequently cause users to avoid formalization.

First, we will discuss the additional effort, or overhead, required of users when they work with formal representations. Second, we describe how systems can end up expecting users to express knowledge which is normally tacit. A third concern

stems from users' reluctance to commit to a structure for evolving or not well understood information. Finally, we describe how useful formal representations vary with the users' situation and may be difficult to negotiate when multiple users are sharing the formal information.

3.1. COGNITIVE OVERHEAD

There are many cognitive costs associated with adding formalized information to a computer system. Foremost, users must learn a system's formal language. Practitioners in some domains use formal languages to describe precisely certain types of information. For example, electricians and electrical engineers use circuit diagrams to communicate circuit designs. However, people seldom use more generic formal languages, such as production rules or frames, for non-computational tasks. While knowledge-based support mechanisms and interfaces can improve a user's ability to use formal languages successfully, Girgensohn's experience (previously mentioned) shows that system concepts related to underlying representations still pose major obstacles for their use (Girgensohn, 1992).

Even if they know a system's formal language, users face a mismatch between their understanding of the information and the system's formal representation; they face a conceptual gap between their own goals and the system's interface. Norman describes the process of bridging this gap or "gulf of execution":

> The gap from goals to physical system is bridged in four segments: intention formation, specifying the action sequence, executing the action, and, finally, making contact with the input mechanisms of the interface (Norman, 1986, p. 39).

As this quote implies, formalisms are often difficult for people to use because they need to take many extra steps (and make additional decisions) to specify anything. These extra decisions may involve chunking, naming, linking, and labeling, where formal languages require much more explicitly defined boundaries, names for subparts, connections between chunks, and labels for such connections than their informal counterparts. The following two experiences point to how such overhead can increase the cost/benefit ratio of use to a point where a system no longer is viable.

The obstacle created by this conceptual gap between users' goals and systems' formal languages was observed in an early prototype of the Virtual Notebook System's "interest profile matcher." The profile matcher would, in theory, enable users of the system to locate other users with certain interests and expertise. The vocabulary used in profiles was the Medical Subject Headings (MeSH), a set of around 20,000 terms divided into about twelve interconnected trees (forming a directed acyclic graph) which is used by medical journals to index articles. Defining an interest profile required choosing terms out of the hierarchies of concepts which best described one's interests. Queries for locating people also required

choosing terms from MeSH terms and attaching "matching ranges" so that all terms in a given range in the MeSH hierarchies would be considered a match. The matching ranges were necessary because MeSH was large enough to experience the vocabulary problem (Furnas et al., 1987) – people using different terms to describe the same topic. With the increase in expressiveness in queries came an increase in difficulty to define queries. Work on the profile matcher was discontinued because the effort required to define interests and queries of sufficient clarity overcame the usefulness of the service the system was to provide.

In an experiment in applying Assumption-based Truth Maintenance Systems (ATMS) derived dependency analysis (described in de Kleer, 1986) to networks of Toulmin micro-argument structures in NoteCards, a similar conclusion was reached: 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 is important to identify assumptions and contradictions. 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 identified and relative truth values had been determined, the user had performed a significant portion of the network evaluation. In this case, the addition processing done by the ATMS mechanism did not compensate the user for the initial effort.

3.2. TACIT KNOWLEDGE

Tacit knowledge is knowledge users employ without being conscious of its use (Polanyi, 1966). Tacit knowledge poses a particularly challenging problem for adding formal structure and content to any system since, by its very nature, people do not explicitly acknowledge tacit knowledge. The problem of tacit knowledge has resulted in knowledge engineering methods aimed at exposing expertise not normally conscious in experts, such as one described by Mittal and Dym:

> We believe that experts cannot reliably given an account of their expertise: We have to exercise their expertise on real problems to extract and model their knowledge (Mittal and Dym, 1985, p. 34).

When such introspection becomes necessary to produce and apply a formal representation during a task it necessarily interrupts the task; the introspection structures and changes it. These changes may be detrimental to the user's ability to accomplish what he or she set out to do.

An example of this interference is McCall's observation that design students have difficulty producing IBIS-style argumentation even though videotapes of

their design sessions show that their naturally occurring discussions follow an IBIS structure (Fischer et al., 1991). McCall also describes a simple physiological example of this interference: When a person is asked to breath normally, their normal breathing will be interrupted. Furthermore, chances are that introspection about what normal breathing means will cause the person's breathing to become abnormal – exaggeratedly shallow, overly deep, irregular.

To develop seemingly natural formalisms, designers may build systems that use representations based on an analysis of user activities, discourse, or documents; these systems are particularly at risk from this type of interference. For example, argument representations are often derived from analyzing naturally occurring argumentative discourse: speech or text is broken into discourse units; the discourse units are categorized according to their functional roles; then the relationship between discourse units is described in general terms. But, as we can see from the IBIS example above, 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.

Formal representations can be specialized to match the user's understanding of their domain and task. Such careful design can reduce the problems of tacit knowledge, but will still influence the outcome of the task, as described by Hutchins et al.:

> While moving the interface closer to the user's intentions may make it difficult to realize some intentions, changing the user's conception of the domain may prevent some intentions from arising at all. So while a well designed special purpose language may give the user a powerful way of thinking about the domain, it may also restrict the user's flexibility to think about the domain in different ways (Hutchins, Hollan and Norman, 1986, p. 108).

Such specialized formal representations are possible for well-defined tasks, but general tasks like analysis and design evolve over time and vary from person to person. The next two sections describe basic problems for determining the user's tasks, and thus the appropriate representations.

3.3. Enforcing premature structure

The process of formalizing information requires one to commit to an explicit structure for the information. One definition of structure is "the elements of an entity or the position of such elements in their relationships to each other." Since a user's understanding of any non-trivial task, such as performing an analysis or completing a design, evolves as they attempt to complete the task, users resist making such commitments. The negative effects of prematurely or unnecessarily imposing a structure have been recorded in both the hypertext (Halasz, 1988) and design rationale (Shum, 1991) literature.

In his studies of how people organized information in their offices Malone found that office workers perceived the negative effects of prematurely structuring

information (Malone, 1983). In particular, one of the subjects in Malone's study said of a pile of papers waiting to be filed:

> You don't want to put it away because that way you'll never come across it again. . . . it's almost like leaving them out means I don't have to characterize them. . . . Leaving them out means that I defer for now having to decide – either having to make use of, decide how to use them, or decide where to put them (Malone, 1983, p. 107).

This quote points out the perception that information formalized incorrectly or inconsistently will be more difficult to use or simply be of less use than information not formalized. This problem can also be seen in the directory structures of UNIX, Mac OS, or DOS users. Many users have large numbers of disassociated files in the top level directory (or folder) of their machine or account. Most of these users know how to create subdirectories or folders to organize their files but postpone classification until they "have more time" or "the mess gets too bad." For these users the perceived benefit or organizing their files does not make up for the effort required to organize the files and the possible cost of mischaracterizing the files.

3.4. Different people, different tasks: Situational structure

The difficulties of creating useful formalizations to support individuals 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 conventions for chunking, labeling, and linking of the information, as well as on the encoding of particular instances. As has been discussed in the context of earlier examples in the use of tools to capture design rationale, the prospects of negotiating how information is encoded in a fixed representation are at best difficult.

Differences occur not just within a group of users but between groups as well. A study of the communication patterns in biomedical research groups showed that the characteristics of the research being performed influenced the organization and communication of the research groups (Gorry et al., 1978). A system which attempts to impose a particular structure on communication will likely not match any given group's actual communication structure.

The problem of situational structure does not arise only when multiple people use the same structure; it can also arise when the user's task changes. The context of the new task may not match the existing structuring scheme. In their list of what are commonly considered the most important properties of a formal system, Winograd and Flores include:

> There is a mapping through which the relevant properties of the domain can be represented by symbol structures. This mapping is systematic in that a community of programmers can agree as to what a given structure represents (Winograd and Flores, 1986, p. 85).

Our own experience seems to indicate that domains for which this is true may be quite small and task dependent. A representation that is suitable for one task may not be appropriate for a very similar related task. For example, a representation developed for the process of assessing foreign machine translation efforts proved to be of limited value in the closely related task of evaluating Spanish-English machine translation software (Marshall and Rogers, 1992). The second task shared source materials with the first task, but the representation did not formalize appropriate aspects of these materials. 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 situations require different user support and thus different formal structures.

4. Approaches to minimizing problems of formalisms

The difficulties of working with formalisms do not have a simple solution. Much like software engineering, where programmers must formally define programs to a computer, there is no single "silver bullet." As Brooks has said of software engineering problems, the interfaces through which formalisms are developed are often part of the problem, but they only contribute "accidental complexity" (Brooks, 1987) to the overall task. Fortunately, unlike software engineering, most of the systems we have discussed do not rely on bug-free formalisms, and thus are amendable to approaches not possible in software engineering.

Although difficulties introduced by formalization are widespread and users are justified in their resistance to or rejection of some formalization tasks, there are viable approaches to this system design dilemma; we describe five of them in this section. (1) Designers need to work with users to reach a shared understanding of the use situation and the representations that best serve it; (2) Designers must identify what other services or user benefits the computer can provide based on trade-offs introduced by additional formalization; (3) Designers should also expect, allow, and support reconceptualization and incremental formalization in longer tasks; (4) Taking a similar, computationally-based approach, designers may provide facilities that use automatically recognized (but undeclared) structures to support common user activities; (5) Finally, while not part of system design per se, training and facilitation can be used to help users effectively work with embedded formalisms.

4.1. IDENTIFYING THE ESSENTIALS FOR TASK

Some information must be formalized for a computer system to perform any task. A work processor must be told the order of characters, a drawing program must be told the color and shape of objects being drawn, and a circuit analyzer must be told the logical circuit design. Interaction based on a limited-domain formalism can become transparent when the user has become skillful in expressing information 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 to support intellectual work and, informationally, what does it require? What must be 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 hypertextual structure need to be specified? To answer these questions, participatory design techniques can be applied to gain an understanding of the users' work practices and possible formalisms to support these practices (Greenbaum and Kyng, 1991).

For example, in Blomberg, Suchman, and Trigg's account of the Work-Oriented Design project, one possible focus for deploying technology in a law firm was document retrieval and reuse. By using non-traditional representations of the attorney's work (like videotape), ethnographers were able to communicate with developers that an attorney in their study relied on page appearance (and not simply on textual content) to identify a desired document in his file cabinet. This in turn suggested that a tool to support his document retrieval and reuse could not depend on content-based representations and retrieval techniques, but rather needed to include appearance-based representations and retrieval methods to be effective for him (Blomberg, Suchman and Trigg, 1994).

Predevelopment ethnographic study can inform software design to a point. Because the introduction or replacement of software changes the associated work practices, systems must be evaluated in real or simulated situations to fully understand these interactions.

4.2. EVALUATING COST/BENEFIT TRADE-OFFS TO SELECT FEATURES

Another approach to addressing the problems of formality is to make some formalization only required for using optional features of the system. Many systems provide functionality which is not necessary for some uses of the system but is available to users who want the added benefits of providing more information. Paragraph styles might be such information in a word processor. A user can accept the default paragraph style to write a paper, and override each paragraph's style with a preferred font and spacing for the individual document elements. He or she would thus never explicitly define the document structure, but would see a similar document appearance. Over time, our hypothetical user might learn to use the feature for defining paragraph style as he or she needs to reformat the document

multiple times. Users can and do learn to use features as their tasks require or as they re-evaluate the cost of not learning a particular feature.

It follows that some such features may be used infrequently. Spreadsheet programs include many features which are used only by a small percentage of the user community (Nardi and Miller, 1990). The rest of the users either get by without using the features or asking for help when they cannot avoid such use. Because system development time and money is limited, designers need to be wary to spending too much time incorporating features which only a small segment of their user community will ever use. In information systems, our experiences indicate that features requiring greater degrees of formality end up being less frequently used.

The experience with Aquanet, discussed in Section 2.1, provides an unanticipated example of this – users decided that formally representing relationships was optional even through this was not the use expected by the developers. Aquanet could still be used for the majority of the intended tasks but was not able to provide certain types of reasoning support which relied on formalized relationships between information objects.

4.3. Gradual formalization and restructuring

Longer tasks necessarily involve reconceptualization; the gradual evolution of human understanding during task performance underlies many of the problems associated with formalization. Providing mechanisms for information to be entered in a less formal representation and then be incrementally formalized and structured is thus a fundamental way system designers can support intellectual activities with computational tools (Shipman, 1993).

Incremental formalization strategies seek to reduce the overhead of entering information, and defer formalization of that information until later in the task. This approach divides up the overhead associated with formalizing information in the system by dividing up the process. Another advantage is that incremental formalization strategies do not require people to impose premature structure when they record information. Like the desks of the office workers in Malone's study (Malone, 1983), information in such systems can be kept without structure until the user wants to add structure.

In the Hyper-Object Substrate (HOS) (Shipman and McCall, 1994) we have investigated the potential to support users by suggesting possible formalizations based on recognized patterns in textual information. In HOS, suggestions for new attributes or relations in the knowledge base were presented to the user for acceptance, modification, rejection, or just to be ignored. Experience with HOS indicates that such suggestions not only reduce the overhead of users providing formalizations, but have the possibility of bringing previously tacit knowledge to consciousness. A similar result is reported in Stevens' account of the use of Infoscope (Stevens, 1993), a system that suggests information filters based on the

users' reading patterns of Usenet News. In Steven's study, a particular suggestion triggered one user to better understand unstated goals and assumptions underlying his Usenet News reading. By helping the user understand their own goals the system helps overcome potential barriers to formalization.

Another example of a suggestion mechanism which helps users formalize structure can be found in VIKI (Marshall and Shipman, 1995), a spatial hypertext system designed to better support the types of non-verbal interpretation seen in Aquanet. In this case, heuristic algorithms are used to find recurring visual/spatial patterns in a layout of information objects; these patterns are indicative of possible relationships among objects. Inferred structures of this sort are used to help users develop representational schemas (the meta-level language of the information space) and to identify specific relationships among groups of objects (like sets and lists).

4.4. Ephemeral structure on demand

Incremental formalization techniques and structure suggestion mechanisms are effective as long as they don't overwhelm a user with too many requests to acknowledge inferred structure. When there is too much inferred structure, a more automated approach is appropriate. Approaches of this sort provide services to the user based on informally represented (i.e. undeclared) information; structures can be inferred through the heuristic recognition of textual, spatial, temporal, or other patterns.

Inferred structure cannot be treated identically to user specified information. One characteristic of possible uses is that the structures are not formalized, but rather used as the basis for interaction. Thus, even if the system's inferences are incorrect at times, as long as they are right part of the time and it is apparent to the user when the system has made the wrong inference, features based on automated recognition of implicit structure will cost the user little for the benefit they provide.

One such use of inferred structure is the hierarchic click-selection feature of VIKI (Marshall, Shipman and Coombs, 1994). In this case the users of VIKI are provided access to the inferred groupings of objects through multiple mouse clicks. The interaction is similar to the expand-selection feature attached to multiple mouse clicks in text editors and word processors. Because VIKI users get visible feedback as to the current selection, incorrect inferences are easily noticed. Another distinctive property of this use of inferred structure is its transience. Because the effects of the inference, the selection of a set of objects, is transient the effects of an incorrect inference do not have an impact on later use of the system.

Similarly, Tivoli, an electronic whiteboard program, uses undeclared structures to support user activities like list and table manipulation (Moran et al., 1994). Once again, interaction is facilitated by system interpretation of layout; the recognition is lightweight, and does not interfere with the normal course of user activities.

4.5. Training, facilitation, and intervention

The approaches we have discussed so far suggest alternatives for reducing use difficulties inherent in systems that use embedded formalisms. Another approach to improving the acceptance of such systems involves helping users learn and understand the expected use of the formalisms through training or through facilitation. Sometimes developers may intervene – at least on a temporary basis – to help users through a difficult portion of the formalization.

As our earlier observations of novice hypermedia users show, the expressive capacity of a system is not necessarily realized intuitively, through use. Instead, to help users learn enough about a system and its embedded formalisms to make effective use of them, training may be necessary and desirable. But training is often insufficient support if the formalisms are complex, or represent a methodology that is far from the users' experience. In these cases, human facilitation has often ensured the success of a system. For example, companies supplying software for recording design rationale find that facilitation is an important part of their business; designers, users, and technology all interact to change practice.

5. Conclusions

We have sought 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 users experience in defining, applying, and instantiating formalisms are not just interface problems. Users are hesitant about formalization because of a fear of prematurely committing to a specific perspective on their tasks; this may be especially true in a collaborative setting, where people must agree on an appropriate formalism and the conventions for encoding information into them. Even when users know precisely what they want to formalize there is the added overhead of learning the formalism and determining how to instantiate their desires in the formalism provided. Additionally, achieving an understanding of what to formalize can require users to become conscious of knowledge that is usually tacit.

There are decisions that system designers can make to reduce the need of formal information by systems and also methods to make it easier for users to provide this information. Designers should observe the current practices involved in the user task to determine what representational features are required and determining what explicit formalization can be asked of users. Systems should be designed to support the process of incremental formalization and structure evolution as tasks are reconceptualized. Finally, systems' designers should determine if it is possible to provide services based on inferred structure in informally represented information.

As groupware and collaborative system designers, it is tempting to add more powerful features that rely on formal information. We must temper that urge and

consider the difficulty that the user will have providing that information before relying on it for the success of our systems.

Acknowledgments

This work was supported in part by the National Science Foundation under grant IIS-9734167. We thank the numerous reviewers whose comments have improved this paper. We also thank Jonathan Grudin and Tom Moran for reading and providing comments on this paper.

References

Bannon, L. (ed.) (1995): Commentaries and a Response in the Suchman-Winograd Debate. *Computer Supported Cooperative Work*, vol. 3, no. 1, pp. 29.

Blomberg, J., L. Suchman and R. Trigg (1994): Reflections on Work-Oriented Design in Three Voices. *Social Science, Technical Systems, and Cooperative Work.*

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

Brunet, L.W., C.T. Morrissey and G.A. Gorry (1991): Oral History and Information Technology: Human Voices of Assessment. *Journal of Organizational Computing*, vol. 1, no. 3, pp. 251–274.

Bullen, C.V. and Bennett, J.L. (1990): Learning From User Experience With Groupware. *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW'90)*. New York: ACM.

Carroll, J.M. and T.P. Moran (1991): Special Issue on Design Rational. *Human-Computer Interaction*, vol. 6, nos. 3–4.

Conklin, J. and M. Begeman (1988): gIBIS: A Hypertext Tool for Exploratory Policy Discussion. *Proceedings of the Conference on Computer Supported Cooperative Work*. New York: ACM.

Conklin, E.J. and K.C. Yakemovic (1991): A Process-Oriented Approach to Design Rationale. *Human Computer Interaction, Special Issue on Design Rationale*, vol. 6, nos. 3–4, pp. 357–391.

de Kleer, J. (1986): An Assumption-Based TMS. *Artificial Intelligence*, vol. 28, pp. 127–162.

Dourish, P., J. Holmes, a. MacLean, P. Marqvardsen and A. Zbyslaw (November 1996): Freeflow: Mediating Between Representations and Action in Workflow Systems. *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW'96)*. New York: ACM.

Ellis, C.A., S.J. Gibbs and G.L. Rein (1991): Groupware: Some Issues and Experiences. *Communications of the ACM*, vol. 34, no. 1, pp. 38–58.

Eriksson, H. (1991): *Meta-Tool Support for Knowledge Acquisition.* Ph.D. dissertation, Linkoping, Sweden: Department of Computer Science, Linkoping University. Linkoping Studies in Science and Technology, Dissertations No. 244.

Fischer, G., A.C. Lemke, R. McCall and A. Morch (1991): Making Argumentation Serve Design. *Human Computer Interaction*, vol. 6, nos. 3–4, pp. 393–419.

Furnas, G.W., T.K. Landauer, L.M. Gomez and S.T. Dumais (November 1987): The Vocabulary Problem in Human-System Communication. *Communication of the ACM*, vol. 30, no. 11, pp. 964–971.

Girgensohn, A. (1992): *End-User Modifiability in Knowledge-Based Design Environments*. Ph.D. dissertation, Boulder, CO: Department of Computer Science, University of Colorado. Also available as TechReport CU-CS-595-92.

Glance, N., D. Pagani and R. Pareschi (November 1996): Generalized Process Structure Grammars (GPSG) for Flexible Representations of Work. *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW'96)*. New York: ACM.

Gorry, G.A., R.M. Chamberlain, B.S. Price, M.E. DeBakey and A.M. Grotto (1978): Communication Patterns in a Biomedical Research Center. *Journal of Medical Education*, vol. 53, pp. 206–208.

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

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

Halasz, F.G. (July 1988). Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems. *Communications of the ACM*, vol. 31, no. 7, pp. 836–352.

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

Hutchins, E.L., J.D. Hollan and D.A. Norman (1986): Direct Manipulation Interfaces. In D.A. Norman and S.W. Draper (eds.): *User Centered System Design, New Perspectives on Human-Computer Interaction*. Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

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

Lee, J. (October 1990): SIBYL: A Tool for Managing Group Decision Rationale. *Proceedings of the Conference on Computer-Supported Cooperative Work*. New York: ACM.

MacLean, A., R. Young, V. Bellotti and T. Moran (1991): Questions, Options, and Criteria: Elements of a Design Rationale for User Interfaces. *Human Computer Interaction*.

Malone, T.W. (January 1983): How do People Organize Their Desks? Implications for the Design of Office Information Systems. *ACM Transactions on Office Information Systems*, vol. 1, no. 1, pp. 99–112.

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

Marshall, C., F. Halasz, R. Rogers and W. Janssen (1991): Aquanet: A Hypertext Tool to Hold Your Knowledge in Place. *Hypertext '91 Conference.*

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

Marshall, C. and F. Shipman (1995): Spatial Hypertext: Designing for Change. *Communications of the ACM*, vol. 38, no. 8, pp. 88–97.

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

Mittal, S. and C.L. Dym (1985): Knowledge Acquisition from Multiple Experts. *AI Magazine*, vol. 6, no. 2, pp. 32–36.

Monty, M.L. (1990): *Issues Supporting Notetaking and Note Using in the Computer Environment*. Ph.D. dissertation, San Diego, CA: Department of Psychology, University of California, San Diego.

Moran, T., P. Chui, B. vanMelle and G. Kurtenbach (1994): Implicit Structures for Pen-Based Systems Within a Freeform Interaction Paradigm Technical Report. 3333 Coyote Hill Road, Palo Alto, CA: Xerox Palo Alto Research Center.

Musen, M. (1989): An Editor for the Conceptual Models of Interactive Knowledge-Acquisition Tools. *International Journal of Man-Machine Studies*, vol. 31, pp. 673–698.

Nardi, B.A. and J.R. Miller (October 1990): A Ethnographic Study of Distributed Problem Solving in Spreadsheet Development. *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW'90)*. New York: ACM.

Newman, S. and C. Marshall (1992): *Pushing Toulmin Too Far: Learning from an Argument Representation Scheme Technical Report*. 3333 Coyote Hill Road, Palo Alto, CA: Xerox Palo Alto Research Center.

Norman, D.A. (1986): Cognitive Engineering. In D.A. Norman and S.W. Draper (eds.): *User Centered System Design, New Perspective on Human-Computer Interaction*. Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

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

Polanyi, M. (1966): *The Tacit Dimension*. Garden City, NY: Doubleday.

Shipman, F. (1993): *Supporting Knowledge-Base Evolution with Incremental Formalization*. Ph.D. dissertation, Boulder, CO: Department of Computer Science, University of Colorado. Also available as TechReport CU-CS-658-93.

Shipman, F., R. Chaney and T. Gorry (November 1989). Distributed Hypertext for Collaborative Research: The Virtual Notebook System. *Proceedings of Hypertext'89* (Pittsburgh, PA). New York: ACM.

Shipman, F. and R. McCall (1994): Supporting Knowledge-Base Evolution with Incremental Formalization. *Human Factors in Computing Systems, INTERCHI'94 Conference Proceedings.* ACM.

Shum, S. (1991): Cognitive Dimensions of Design Rationale. In D. Diaper and N.V. Hammond (eds.): *People and Computers VI*. Cambridge, UK: Cambridge University Press.

Stevens, C. (1993): *Helping Users Locate and Organize Information.* Doctoral dissertation, Department of Computer Science, University of Colorado.

Suchman, L.A. (1987): *Plans and Situated Actions*. Cambridge, UK: Cambridge University Press.

Suchman, L. (1994): Do Categories Have Politics? *Computer Supported Cooperative Work*, vol. 2, no. 3, pp. 177–190.

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

Waterman, D.A. (ed.) (1986): *A Guide to Expert Systems*. Addison-Wesley.

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

Winograd, T. (1994): Categories, Disciplines, and Coordination? *Computer Supported Cooperative Work*, vol. 2, no. 3, pp. 191–197.