Hypertext for Software Engineering

Walt Scacchi, Institute for Software Research, University of California, Irvine
May 2001

To appear in, J Marciniak (ed.), *Encyclopedia of Software Engineering, 2nd. Edition*, John Wiley and Sons, Inc., New York, 2001

INTRODUCTION

What is the power of hypertext in supporting software engineering? (The term "hypertext" is treated throughout as synonymous with "hypermedia" which denotes the nonlinear representation of interrelated textual, graphic, filmic, or auditory information.) Power in other contexts usually refers to the ability of some entity or agent to affect the behavior of another or to achieve advantage over another in ways that the other cannot avoid. Following this, one might expect that as a *technology*, hypertext systems offer relative advantage over alternatives to automated text/document information systems, including conventional word processing systems, file systems, and database management systems.

Hypertext systems offer a degree of information processing power that enables new kinds of applications, much like the advent of expert system shells enabled the creation of expert applications for domains such as computer configuration and diagnostic pathology. (Such shells are considered "domain-independent," but when instantiated with knowledge or rules about patterns among application domain data, the expert system application is considered "domain-specific.") Similarly, as a *medium* for writing and reading electronic documents hypertext systems allow the redefinition of the structure and content of documents that alter the constraints and opportunities for conveying information in contrast to the linear print medium. Thus the power of hypertext in whatever form it exists is subtle and incremental; it is potential rather than kinetic. However, such power can accumulate in different application domains in the long run if effectively mobilized and integrated with domain-specific tasks, processing mechanisms, and workflows.

The power of hypertext technology in different application domains can be realized in a number of ways. In particular, technology that supports a unified view of the hypertext documents, production processes, processing mechanisms, and settings of use appears to offer the greatest power at this time. To see this, the concept of a domain-specific hypertext environment (DSHE) is defined in the following section. Next, the domain of software production is examined in order to identify how hypertext tools and document production/consumption techniques can play a role to facilitate software engineering objectives. The focus then shifts to describe the structure of information to evolve within a DSHE in terms of product, task, and setting information for the domain of software production (Anderson 1999; Cybulski and Reed, 1992; Garg and Scacchi, 1989,1990; Oinas-Kukkonen 1999; Noll and Scacchi 1991,1999; Roth, Aiken, and Hobbs 1994; Scacchi, 1989,1991; Wiil, Nurnberg, and Leggett 1999). Together, these sections then set the final stage for a discussion of the power of DSHE. Subsequently, seven ways the power of DSHE can be realized are identified, followed by an examination of the costs that are borne in order to realize this power.

DOMAIN-SPECIFIC HYPERTEXT ENVIRONMENTS (DSHE)

Until about 1985, the number of automated hypertext systems could be easily counted. Since then, interest in the development and use of hypertext systems has exploded. Conklin (1987) reviews most of what might be called the first generation of hypertext systems in his survey. He identifies a dozen features that characterize the 18 hypertext systems he surveys. The *ACM Hypertext Conferences*, *Computer-Supported Cooperative Work* conferences, and the *ACM Conferences on Information Systems (SIGGROUP)* have been showcases for more recent progress in the research and development of hypertext systems. However, until recently few of the new hypertext systems provide support for all of the features Conklin identifies.

Domain-Specific Hypertext

Many of the published descriptions of hypertext systems only make casual reference to the contents or nonlinear structures of their hypertext bases. Instead, their focus is usually on the computational mechanisms through which a subject hypertext is processed. In most cases, this is reasonable since the research publications seek to describe innovative technologies that might be applied to many different hypertext application domains. Hypertext mechanisms are also becoming an increasingly popular means for interactive delivery of on-line manuals or "help" information in a growing number of commercial software products. However, the focus here is to examine on more of an equal footing how the content and structure of an application domain goes along with the use of different processing mechanisms and technologies. This is a way of saying that precollege students, humanities scholars, technical professionals, periodical publishers, insurance actuaries, manufactured product designers, and software engineers all process different kinds of information, in different media, in different settings, with different processing needs and infrastructure support. Alternatively, this is another way of saying what has already been realized elsewhere: in order to effectively support the production and consumption of complex interactive documents, one should choose information structures and processing mechanisms that fit the users' skills, tasks, and resources in the application domain. This choice is preferred because if the processing mechanisms are inappropriate for the application domain, they will fit poorly, causing the system to be more troublesome to use, more costly to maintain, more likely to be resisted, and most likely to fail (Kling and Scacchi, 1982; Bendifallah and Scacchi, 1987). Thus what makes a hypertext system domain-specific is the codification and configuration of the application domain's agents, objects, attributes, relations, constraints, tasks-processes, transaction event rules, mechanisms, and resources into the information structures that are woven together to form its hypertext.

Hypertext Environments

Environments represent another new direction for investigating the information-processing capabilities of hypertext systems. Environments represent coupled ensembles of automated systems and techniques for their use by people with skills and interests appropriate for their application domain. *Hypertext environments* represent the integration of hypertext features such as those described in Conklin's survey (hierarchical structures, nonhierarchical links, typed links, procedural attachment, concurrent multiuser support, graphical browsing, and so forth) with

automated tools, processing tasks, and application workplaces as a way to organize and access large databases of nonlinearly structured text or on-line media. The combination of domain-specific hypertext information structures with application processing environments leads to an infrastructure technology for different kinds of application domains called *domain-specific hypertext environments.*

APPLICATION DOMAINS FOR HYPERTEXT ENVIRONMENTS

One application domain for hypertext environments will be examined in order to identify and characterize its processing mechanisms and information structures. Possible application domains could include encyclopedic and classical studies, creative writing and interactive fiction, journal publication, insurance policy processing, software production, or others (Scacchi, 1989). The choice here is to examine software production as an application domain for hypertext environments. Thus this examination will also serve to identify the various kinds of information structures, processes, and settings being used in domain-specific hypertext environments for software production.

Software Production

Software production is concerned with the development, use, and evolution of large software system applications and environments. "Large" refers to the fact that such software systems are often developed by teams of 10 to 100 or more software engineers who work on development projects that take months to years to complete, and whose resultant programs are usually in the range of 50,000 to 500,000 source code statements (between 1,000 to 10,000 single-spaced pages). Further, due to the complexity of these programs, it is widespread practice to develop a series of *interrelated multiversion documents* that describe the software production life cycle—the requirements, functional specifications, prototypes, designs, test plans, implementations, user manuals, and maintenance guides—for the system being developed. These documents often represent 5 to 10 times the volume of the source code. As such, it is more efficient for the structure and to some extent the content of these documents to conform to company, industrial, or national standards in order to facilitate parallel/multiperson development, review, and quality assurance.

Software systems are documented using fully structured descriptions (functional specifications, operational prototypes, designs, and source code) whose syntax and semantics can be formally defined and automatically analyzed, and weakly structured descriptions (narrative requirements, user manuals, test plans, and maintenance guides). The contents of these weakly structured documents can be text-processed and understood by people, but because they are not formalized, these documents are often ambiguous and incomplete (Garg and Scacchi, 1990, Roth, Aiken, and Hobbs 1994). As such, automated mechanisms should be provided to *identify* and *trace* relationships across multiple semistructured descriptions of the same system in order to *configure, validate,* and *maintain* the consistency of interrelated software descriptions as they evolve (Narayanaswamy and Scacchi, 1987; Noll and Scacchi 1997; Choi and Scacchi, 1998).

Large software engineering projects produce encyclopedic volumes of semistructured and interrelated descriptions. As such, the production of software life cycle documentation can

represent a substantial fraction of the system's development cost. However, such cost is often necessary because the large scale of the system development effort often implies that the system's documentation will be the focal medium for coordinating the engineering tasks and information flows among developers working at different times and places. Software development requires the substantial coordination of diverse specialists, automated tools, multiple system descriptions, and other organizational resources. Electronic mail and bulletin boards are increasingly essential but usually lack knowledge of the software development products, processes, workplaces and their interrelationships. However, a new generation of automated software engineering technologies are beginning to appear that represent and manipulate these kinds of knowledge (Garg and Scacchi, 1990; Johnson and Erdem 1997; Mi and Scacchi, 1990; Lowry and McCartney, 1991; as well as recent conference proceedings and the journal for automated software engineering). Similarly, to increase the rate and quality of software system production, other technologies including intelligent message management systems (Malone and co-workers, 1987), and on-line catalogs or repositories of reusable software components are being investigated.

Finally, it is interesting to observe that the kinds of information structures and processing mechanisms used for a software production hypertext are generally a superset of those appearing in other domains (Scacchi, 1989). This of course does not imply such a covering with respect to the document content or professional expertise of the other domains. Nonetheless, one can use this domain to examine how the structure of information in a software hypertext is organized and how the attendant processing mechanisms are organized into comprehensive DSHE in the sections that follow.

THE STRUCTURE OF INFORMATION IN SOFTWARE PRODUCTION

Within the domain of software production, documentation is a primary record of engineering activities. The document *nodes* produced are interrelated through various kinds of *links* as a web or semantic network (Quillian 1968). Documents are structured into development folders or *partitions* associated with particular *system/subsystem projects* and *associated staff*. Each partition may contain a standard set of chapter, sections, and subsections that store and organize software product descriptions. These can be represented as skeletal *indices* that organize collections of related descriptions as *nodes* of the software hypertext. Further, as the volume of documents produced is often quite large, additional document structuring mechanisms are needed to provide views that cut across standard document structures. Such views constitute interlinked document *compositions.* These compositions are used for selecting partitions, indices, or nodes that can be structured and related following standard linking schemes or ad hoc, user defined schemes. As such, compositions can provide crosscutting views of software documents that are useful for distribution, quality assurance review, browsing, or simply printing an individual's project document contributions. Next, the development, use, and evolution of large software system descriptions can be explicitly organized and represented in terms of the *tool integration*, *processes* and *settings* where they are produced. Last, the structure of information within the domain of software engineering defined in these terms accommodates a rigorous formalization of the underlying abstract relationships that are embedded in a software hypertext (Garg, 1988).

In the remainder of this section, the information structures highlighted in the preceding paragraph are elaborated, based on experiences with software hypertext environments in the System Factory Project at USC (Garg and Scacchi, 1988,1989,1990; Noll and Scacchi, 1991, Scacchi 1991) and elsewhere (Anderson 1999; Cybulski and Reed, 1992; Fielding and co-workers, 1998; Oinas-Kukkonen 1999; Whitehead 1997; Wiil, Nurnberg, and Leggett 1999).

Software Hypertext Information Structures

Links. Links represent relations (Woods 1975) that identify specific associations within or between software descriptions. Keywords and annotations are special kinds of links that are often supported with predefined processing mechanism. Keywords and user-defined links may serve as simple pointers for tracing the occurrence of designated terms across and between software documents. Annotations can represent hierarchically linked narratives that further describe the designated term or object to indicate its meaning, explanation, or history of its use; or decisions pertaining to its definition and purpose. Users can also define links as semantic relations which in turn can be static or operational. Static links denote a simple logical relation between linked software descriptions, whereas operational links imply some standard or user-defined computational processing that is automatically invoked when linked items are visited or modified. In general, as all links indicate some kind of relation, they can be stored and managed by a relational or object-oriented database management system (DBMS) (Narayanaswamy and Scacchi, 1987; Garg and Scacchi, 1989,1990; Choi and Scacchi, 1991,1998; Wiil 1995: Wiil, Nurnberg,and Leggett 1999). In this way, linked software object descriptions can be indexed, browsed, and relationally queried through the mechanisms of the DBMS, including query processors, report generators, pattern matchers, and fourth generation languages. Further, as a secondary benefit, these search and navigation mechanisms help to reduce or eliminate the chance of getting lost or disoriented when traversing the hypertext (Garg and Scacchi, 1990), which was a common problem with the first generation hypertext systems (Conklin, 1987).

Hypertext Nodes. Nodes denote semi-structured object descriptions of various length and substructure (Malone and co-workers, 1987; Garg and Scacchi, 1990). Semi-structured indicates that object descriptions are structured to some degree—that is, all objects possess descriptive attributes that characterize the structure, content, and purpose of the object. The range of possible values these attributes take on may be either completely defined (for example, by formal language specification or enumeration, as in Garg 1988) or weakly defined. However, the presence and formalization of these attributes determines the degree to which the descriptions can be parsed, analyzed, and interpreted by processing mechanisms. Processing tools like language-directed editors, application generators, specification and design analyzers, and compilers to determine their traceability, consistency, completeness, or correctness (Garg and Scacchi, 1990; Narayanaswamy and Scacchi, 1987; Choi and Scacchi, 1998). Subsequently, nodes and forms provide a common, persistently typed substrate for representing and managing software descriptions in ways that facilitate CASE tool integration.

Indices. In simple terms, indices represent aggregations of nodes. For example, individual nodes might be used to denote the sections and subsections within the index of a Requirements Analysis document for a project. More generally, indices can be used to define the structure and type of

software system information organized as a collection of life-cycle document structures that can be standardized and reused across many project teams of software engineers. This standardization can therefore support parallel authorship of multiple software documents when the interrelationships between documents are made explicit *a priori*. This arrangement thus creates opportunity for improving the coordination and productivity of teamwork within a system development effort.

Compositions. These are threaded networks of indices and/or nodes that denote some user-defined association (Garg and Scacchi, 1988; Fielding and co-workers, 1998). For example, configurations can be composed for printing only the sections or subsections of an article modified by an author since the last revision. In multi-authored documents such as those associated with large software systems, this is a particularly useful feature that facilitates coordination and project status monitoring. Similarly, compositions allow linked software descriptions to be managed, viewed, or evolved in ways that support software architecture design (Fielding and Taylor, 2000) and software configuration management (Narayanaswamy and Scacchi, 1987; Noll and Scacchi 1997; Choi and Scacchi 1998). Finally, compositions may be employed as a scheme for defining and managing versions of shared collections of nodes or indices in order to facilitate distributed authorship (Noll and Scacchi 1999; Whitehead and Wiggins, 1998)

Partitions for Projects and Teams. Partitions provide a semantic structuring mechanism (Sacerdoti 1977) whereby collections of hierarchical indices of nodes can be composed, standardized, and shared by classes of users. For example, in the System Factory Project, partitions were organized by work group, project, or project site (Scacchi, 1991; Noll and Scacchi, 1991). Partitions represent contexts (Delisle and Schwartz, 1987, Sacerdoti 1979) for structuring access to a hypertext of software object descriptions. Thus individual index or node instances cannot appear in more than one partition class. However, people working within a partition may browse, link, or compose across multiple partitions. This supports the assignment of standard node/index processing tools or applications to designated types of software descriptions (Anderson, 1999; Anderson and co-workers, 2000; Garg and Scacchi, 1990, 1989; Noll and Scacchi, 1991,1999). Similarly, it also helps to minimize the chance of getting lost in a software information hyperspace because a user always works within a known context with defined indices and nodes.

Tool Integration. The preceding information structures can be employed to represent and associate documents and artifacts that emerge during software production. However, different types of documents and artifacts may be associated with specific types of tools. Thus, hypertext support for software engineering requires a capability to associate and integrate different kinds of software engineering tools or other application programs. Part of the challenge of tool integration includes providing alternative integration schemes, interfaces, or mechanisms in order to accommodate the heterogeneity of tools, nodes, relations, and processes that are found in software production (Anderson 1999; Anderson and co-workers, 2000; Noll and Scacchi 1999; Whitehead 1997; Wiil 1995). Another part of the challenge is accommodating the transition from the integration of tools to the integration of components and services (Nurnberg 2001; Wiil,

Nurnberg, and Leggett 1999), as well as process (Mi and Scacchi 1992; Noll and Scacchi 1999,2001).

Software Engineering Processes. The preceding information structures are used to describe the organization of software object descriptions as products that document the software life cycle. However, it is often the case that the software life-cycle process—the sequence of tasks and associated processing mechanisms that create and manipulate software descriptions—is itself subject to alternative definitions and task compositions. Thus the tasks that software engineers perform should also be developed and organized in ways analogous to other software object descriptions.

The information structures for specifying software engineering tasks need to describe the sequences of processing mechanism invocations that simplify the routine manipulation and interchange of software documents. Consider source code program documents for example (Noll and Scacchi 1999): Here the processing mechanisms for manipulating source code descriptions include editors, compilers, debuggers, program linkers and loaders, as well as formatters for displaying highlighted ("pretty-printed") program listings. The structure of the task of developing a working program usually requires the use of each of these mechanisms. The sequence of their invocation is mostly nondeterministic and nonprocedural, but easily tracked by individual software engineers for small programs. However, if the programs being developed are large, built by teams developing multiversion program components according to an elaborate lifecycle engineering methodology, then the program writing task becomes complex and costly if not well-coordinated (see also PROCESS MODELS IN SOFTWARE ENGINEERING).

The description of software object/document processing tasks can be decomposed at many levels of detail and interrelationship. First, there must be support for specifying the task- of task specification. This metatask description is needed for organizing tasks, specifying their interrelationships, and assigning them to appropriate partitions. Metatask descriptions must be maintained because the content and structure of task descriptions evolve in workplaces in unexpected ways (Bendifallah and Scacchi, 1987, 1989, Mi and Scacchi, 1993). Second, there are two classes of document tasks for which many subtypes can be identified. These are *management tasks* of project administrators, and *engineering tasks* of technical project staff. Management tasks focus on activities such as decomposing system development projects into subsystems, assigning staff and processing mechanisms, scheduling and budgeting subsystem description development, monitoring project progress and productivity, acquiring and maintaining an adequate supply of staff and computing resources, assuring the quality of the integrated and validated final product document assembly, and redoing any of these when things break down, foul-up, or when external conditions dictate (Scacchi, 1984). Engineering tasks are performed by individual or small groups of engineers who create, manipulate, and interchange component documents assigned to them, among other things (Garg and Scacchi, 1990, 1989; Bendifallah, 1989). These tasks include analyzing system requirements, developing functional specifications, software prototyping and design, coding and testing programs, maintaining existing systems, and so forth.

One should recognize that the management and engineering tasks are interrelated in many ways. For example, in order to validate that a system is fully operational and ready for delivery, the task is one of assuring that

- All the right versions of;
- All of the right system components were selected and;
- Composed in the right order and;
- Appropriately tested.

Each of these subtasks in turn requires management subtasking. For example, each of these subtasks assumes that the required source code programs were developed and run through the assigned set of source code processing mechanisms. However, they also assume the existence of either an automated processing mechanism or (manual) administrative mechanisms for checking to see that the components fit together in a consistent and complete manner. When the number of components is in the hundreds or thousands, each existing in one or more version, each potentially fitting into many alternative configurations, and each having one or more sets of data for testing, then a combinatorial explosion of alternatives must be engineered, configured, and managed. The complexity of the emerging system begs for coordination and automation rules that simplify and maintain a closed system description whose consistency and completeness can be directly assessed. But the complexity of system artifacts emerging from the development effort within an embedding environment requires that the successful performance of the management and engineering tasks will be interdependent.

All tasks, regardless of level of description, describe a potentially nonlinear sequence of activities called *task chains* (Kling and Scacchi, 1982; Gasser, 1986; Bendifallah and Scacchi, 1987, 1989; Garg and Scacchi, 1989; Mi and Scacchi, 1990). These activities affect some concrete or abstract transformation of a software node, index, composition, or partition. For example, in the task of implementing a software system design as a program, the creation of a successfully compilable program component is a necessary action. Other steps in the sequence include

- Understanding the software design;
- Developing a program implementation strategy;
- Establishing which processing mechanisms are available;
- Debugging an anomalous program behavior, and so forth.

In turn, tasks can be further decomposed into subtasks and eventually into *atomic actions.* The actions represent commands issued in dialogues between an individual and the current processing mechanism in use. For example, understanding the software design can entail

- Browsing a design document;
- Following embedded cross-reference links;
- Querying the design dictionary for information about a particular software object;
- Tracing design details back to the originating system requirements;
- Searching for an existing program component which performs a similarly designed computation, and so forth.

As before, the sequences of actions are also nonlinear or partially ordered (Garg and Scacchi, 1989).

Overall, actions, subtasks, tasks, and metatasks must be articulated, aligned, and coalesced vis-a-vis one another when a system of software life-cycle documents is produced. This can become an emerging, open ended activity that we seek to structure, control, and close so that we can assure its consistency and completeness. As such, these software process descriptions at varying levels of detail can not be guaranteed *a priori* to be consistent, complete, or correct under all possible performance circumstances. Hence the need arises for viewing the creation and manipulation of software process tasks as (semi-structured descriptions that should be managed as domain-specific hypertext (Garg and Scacchi, 1989; Noll and Scacchi 2001; Scacchi, 1989,1991).

Software Production Settings. As previously noted, the workplaces for using domain-specific hypertext include people with different skills, processing mechanisms, and a variety of shared organizational resources. Furthermore, the workplaces for software production are becoming increasing distributed and global, due to the existence of the Web (Fielding and co-workers 1998; Noll and Scacchi 1997,1999,2001). Thus this information can also be described and linked into the software hypertext work environment. For brevity, the focus here is limited to the project participants.

People in software engineering projects are typically divided in terms of management, engineering, customer, and maintainer skills. Further decompositions into subtypes for each can be identified. For management skills, one sometimes sees specialists for process management, quality assurance, scheduling and budgeting, and configuration management. On some projects, these skills might be possessed and put into practice by a single person. However, on large projects divided into many subsystems and small group teams, then these management skills will often be shared among a few key people. Similarly, on the engineering side, there may be specialists for each software life cycle activity, or some subset of life cycle activities. For example, the principal software engineer and a small group of trusted senior engineers may be primarily responsible for defining the overall software system architecture, major subsystems, as well as specifying their operational requirements. These specialists, however, will usually not be found with the additional responsibilities of performing the detail design and implementation of source code modules. (One possible exception to this might arise for modules designated as critical to the overall system performances or integration. But in most projects this will be uncommon.) Similarly, the majority of system programmers will not have the skills for producing a viable set of system requirements or overall design.

However, there are also other people whose participation and skills can affect a software development project. These would be the end-users, or clients, who typically will define the system's general requirements, and the system's maintainers. If the customers have a history of experience in specifying or using diverse software applications, their skills in specifying system requirements will be different from others acquiring or using an unfamiliar application technology. These latter types of customers may, as a result of their inexperience or uncertain knowledge,

frequently change the specification of their requirements. However, it is widely recognized that changing system requirements is one of the most frequent causes of projects being late, overbudget, or otherwise a technical failure or maintenance nightmare. This brings up another class of project participants, software system maintainers.

For a large software system, maintenance activities go on for years. Maintenance tasks—adding functional enhancements, resolving anomalies, tuning system performance, or migrating the system to other platforms—are divided among staff according to subsystem responsibilities. Software maintainers for large systems are generally not the same people who originally developed the system. As such, their knowledge of the system's operational behavior, function, and structure must be derived from either the existing software object descriptions, informal conversations with others, or direct experience. One frequent problem here is that the existing descriptions (source code versus system design) are typically inconsistent, incomplete, or otherwise inadequate (Bendifallah and Scacchi, 1987; Choi and Scacchi, 1998). Thus software maintainers are often at a disadvantage in keeping operational systems viable, unless there are automated maintenance support systems to assist them (Narayanaswamy and Scacchi, 1987; Choi and Scacchi, 1991,1998), or the available software object descriptions were engineered and maintained throughout the project up to this point. As a result, the success of the maintainer's tasks depends on their ability to accommodate or negotiate alternative definitions of their tasks or work arrangements (Bendifallah and Scacchi, 1987, 1989; Mi and Scacchi, 1990).

As such, there are four classes of participants for software production—managers, engineers, customers, and maintainers—that can be further decomposed into task specialists who are interrelated and interdependent. However, their task-skill combinations are constrained by the limits of the organizational resources and automated processing mechanisms allocated to their interlinked project partitions. Thus, the structure, content, and flow of project participants' task-skill organization should be described and managed as evolving software object descriptions (Garg and Scacchi, 1989; Mi and Scacchi, 1990, 1993; Noll and Scacchi 2001; Scacchi and Noll 1997).

Last, it is also important to delineate the structure of processing mechanisms and organizational resources in term of their compositions of class-subclass hierarchies as well as their links to other setting, process, and software object description structures. These are discussed elsewhere (Garg and Scacchi, 1989; Noll and Scacchi, 1999,2001; Scacchi, 1989; Scacchi and Noll 1997).

THE POWER OF DOMAIN-SPECIFIC HYPERTEXT TECHNOLOGY

Given this examination of DSHE, seven dimensions can be identified along which the potential power of DSHE is realized. However, the acquisition, exercise, and accumulation of such power comes with varying costs. Similarly, it is often the case for software systems that their productivity benefits are overestimated, while their costs are underestimated (Kling and Scacchi, 1982). Thus these benefits and costs should be examined.

Productivity Enhancement. DSHE pose many opportunities for enhancing the productivity of their users in different domains. For example, DSHE support explicit articulation of an

application domain in terms of hypertext content structures. Further, DSHE can also organize processing mechanisms, processing tasks, and settings. These provide a framework for organizing information processing work in a systematic rather than ad hoc manner. Standardized document contents and structures support the cataloging and reuse of information across different application instances. Formalization of the hypertext documents, processes, and settings provides a basis for use of knowledgebased processing mechanisms. Hypertext can serve as an integrating medium for coordinating the interconnections and interfaces among multi-author documents. Similarly, DSHE can utilize cooperation and composition mechanisms that allow multiple authors to work in parallel, as well as across different document versions (Whitehead and Wiggins 1998).

Improved Quality of Document Production and Consumption. The "look and feel" qualities of documents increase, especially as both the writers and readers acquire skills and perform tasks that increasingly necessitate the use of hypertext environments in their work. This may be especially true in domains where hypertext technology serves as a groupwork support system, where such systems have been found to increase the participation of "writers" and increase the quality of decisions for action by "readers" (Applegate and co-workers, 1986). Similarly, standardized content structures streamline the composition and binding of multicontribution documents. In turn, interactive readers of such documents can acquire skills with processing mechanisms that exploit available structures or contents. Finally, when document contents and structures can be formalized, automated mechanisms can be employed to check on the consistency and completeness of domain documents as they are created or modified and then prevent or report errors when they are detected.

Hypertext as a Medium for Coordinating Work. When DSHE allow explicit representation and modeling of the relationships among documents, processes, and workplace, automated mechanisms that integrate, manage, communicate, transform, or rapidly recompose product-task-setting information can be introduced. Similarly, when combined with an intelligent message/mail management system (Malone and co-workers, 1987; Garg and Scacchi, 1989), a DSHE can support mechanisms for allocating, collecting, and integrating documents that are interchanged among collaborating workers; as well as tracking, triggering, and filtering documents whose production was scheduled and budgeted (Mi and Scacchi 1990; Noll and Scacchi 2001).

Active System Participation. As hypertext systems become increasingly intelligent and process based, such systems can be empowered to act not merely as passive information repositories but also as active information gatherers or providers. When process knowledge pertaining to the information structures in some document can be developed with automated mechanisms, a DSHE begins to move beyond passive storage and user-directed retrieval capabilities. When information pertaining to knowledge of the domain-specific document structures and contents, the processes of their production and use, and the workplaces where such documents and processes are articulated can be acquired through interaction with system users, then DSHE will move toward becoming active agents capable of producing and using hypertextual information content and structure (Garg and Scacchi, 1989; Johnson and Erdem 1997; Mi and Scacchi, 1990; Scacchi 2000).

Reinforcing Biases and Structured Perspectives. Structured views of large bodies of textual information enable the view developer to try to focus the attention of the reader to particular "points of interest" or connections within the text. These view structures may be intended to direct or persuade the reader of the appropriateness of the viewmaker's interpretation of the importance or meaning of the underlying text. This is familiar to readers of used textbooks that are marked with highlighter pens. It is often easy to have one's attention drawn to the marked passages and accordingly difficult to completely ignore or overlook these same markings. Therefore, the viewmaker can impose a set of standard ways for producing and consuming information structures explicitly represented in an application hypertext and thus encourage "shared" interpretations of its meaning.

Enrichment of Information Processing Skill and Work. For some people the use of DSHE for text searching, browsing, query processing, and other actions is intrinsically motivating or stimulating (Malone and co-workers, 1987; Scacchi 2000; Scacchi and Noll 1997). When groups of these people use DSHE in their collective work, they may then learn how to more effectively coordinate the production of multiauthor documents. Further, the acquisition of hypertext creation and manipulation skills will help create new career and professional job opportunities for those people who first seek to master such systems within their domain of expertise.

Competitive Advantage through Technological Innovation. Employing hypertext systems in particular application domains may realize a competitive advantage to its organizational users (Porter, 1980). For example, the use of new technologies in academic disciplines to study either established research problems in alternative ways or to open up new problem domains is a frequent subject of scholarly research publication. Researchers who have access to these new technologies can realize a new opportunity to extend their line of research work by publishing articles in professional conferences or archival journals that describe their experience in developing, using, evolving, or evaluating hypertext (or Web) mechanisms (Fielding and co-workers, 1998). Most of the scholarly publication outlets are highly competitive regarding articles accepted for publication. The use of new technology does not guarantee acceptance of subsequent research publication. But the opportunity to innovate with hypertext environments appropriate for other academic or commercial domains exists for those capable of exploiting its technical and professional potential.

The Cost of Power: The Hypertext Package

DSHE represent more than merely a set of programs. Instead, DSHE represent a set of requirements for combinations of computing hardware, software, communications interfaces, mass storage systems, as well as people with skills to develop, use, and evolve hypertext applications, policies, procedures, budgets, and schedules for efficient utilization of DSHE in regular information work activities. Unless these requirements are adequately satisfied and sustained, the power of DSHE will not be realized to its full potential. Thus these organizational requirements are referred to as the *hypertext package* (cf. Kling and Scacchi, 1982; Scacchi, 1984).

The potential power of DSHE comes at a price. The cost can be expressed or evaluated in terms of the technological and social resources that get consumed, allocated, or displaced to accommodate the use of DSHE. However, a favorable power-cost ratio can more likely be realized when a DSHE is managed as a package with implicit trade-offs rather than as application programs with instruction manuals.

Consider the following trade-offs: First, the potential for productivity enhancement can emerge from articulate information work structures and integration mechanisms that enable a higher level of parallel activities. However, this could decrease "hidden" pockets of discretionary or slack resources that are considered inefficient but are potentially very effective in mitigating workflow failures or breakdowns (Bendifallah and Scacchi, 1987,1989; Mi and Scacchi, 1993). Second, the potential of hypertext systems as a coordination mechanism depends on the continuity of staff participation in its use. What happens to people not on-line, or who drop off-line for unpredictable periods? Coordination of work in these situations must occur within in the workplace in an ad hoc manner to be realized. Third, skill, work, and professional enrichment may emerge from the mastery of hypertext production and manipulation. However, as with the situation for text processing systems, many technical professionals have broadened their skill base downward with the assimilation of more clerical responsibilities. Similarly, many clerks and typists have broadened their skills upward by acquiring computing specialist skills needed for the correct and sustained operation of office information systems. For clerical staff, upward skill mobility can translate into promotion, salary increase, new career opportunities, or increased workload. On the other hand, for technical professionals expanded skills enable increased workloads and more control over document production but little or no new upward mobility. Will hypertext follow text processing in this regard?

Each of these sample trade-offs again underscore that DSHE should be recognized and managed as a package of information work products, processes, and settings rather than simply a program that comes in a box ready for use on a personal work station. The potential power as well as the costs of DSHE lies within the hypertext package.

CONCLUSIONS

Five observations can be drawn from this discussion on domain-specific hypertext environments:

First, DSHE represent a new generation of organization information infrastructure. As such, investment into their development and evolution should be long term. This will lead to a realization of their maximum potential power but at the same time make clear the costs to be borne to make this infrastructure effective and efficient.

Second, the power of DSHE is derived through structuring hypertext applications to fit with the documents, processing mechanisms, tasks, settings, and their interrelationships specific within a work domain. The power is not inherent in the hypertext system technology. Thus it might be reasonable to expect that people who choose to treat hypertext more as a tool than a package might have more difficulty in understanding or affecting how DSHE work.

Third, DSHE must support the information work for many kinds of producers and consumers. This becomes clear from examining and comparing DSHE in domains of classical studies, creative writing, journal and book publication, insurance policy management, and software production (Scacchi, 1989). Each kind of user may require a distinct viewpoint of the hypertext information, as well as process information with different mechanisms according to different information task sequences. In addition, application domains differ in how hypertext environments and processing mechanisms are structured and standardized.

Fourth, it is instructive to view DSHE as operational information structures that focus on producing interlinked document compositions by reproducing the structure, content, and flow of information work in an application workplace. As a result, attention is not limited to focusing only on the technical detail underlying user interface display, authoring issues, alternative retrieval algorithms, hypertext system implementation issues, and so forth. These are important contributions being investigated by others. This article instead focused on a specific application domain and task environment.

Last, topics for further investigation have been identified in this article. These include

- identifying alternative techniques for integrating message management systems and knowledge-based processing mechanisms into hypertext systems;
- scaling-up hypertext environments to support projects or organizations with hundreds to thousands of users;
- how to increase the strategic value and competitive advantage attributed to DSHE;
- how to determine the requirements for other DSHE; and
- identifying strategies that maximize the power of DSHE that minimize adverse costs.

BIBLIOGRAPHY

K. M. Anderson, "Supporting Software Engineering with Open Hypermedia," *ACM Computing Surveys*, 31(4es), December 1999.

K.M. Anderson, R.N. Taylor, E.J. Whitehead, Chimera: Hypermedia for heterogeneous Environments," *ACM Trans. Info. Systems*, 18(3), 211-245, July 2000.

S. Bendifallah and W. Scacchi, Work Structures and Shifts: An Empirical Analysis of Software Specification Teamwork," *Proc. 11th. Intern. Conf. Software Engineering, ACM*, 1989, pp. 260-270.

S. Bendifallah and W. Scacchi, "Understanding Software Maintenance Work," *IEEE Trans. Software Engineering* 13(3) 311-323, (1987).

S.C. Choi and W. Scacchi, "Formalization and Tools Supporting the Structural Correctness of Software Life Cycle Descriptions", *Proc. IASTED Conf. on Software Engineering*, International

Association of Science and Technology for Development (IASTED), Las Vegas, NV, 27-34, October 1998.

S.C. Choi and W. Scacchi, "SOFTMAN: An Environment for Forward and Reverse Computer-Aided Software Engineering," *Information and Software Technology*, 33(9), 664-674, Nov. 1991.

J. Conklin, "Hypertext: An Introduction and Survey," *Computer*, 20(9), 17-31, 1987.

J. L. Cybulski and K. Reed, "A Hypertext-Based Software Engineering Environment," *IEEE Software*, 9(2), 62-68, 1992.

N. Delisle and M. Schwartz. "Contexts—*A* Partitioning Concept for Hypertext," ACM *Trans. Office Info. Systems* 5(2), 168-186, 1987.

R. Fielding, E.J. Whitehead, K. Anderson, P. Oreizy, G. Bolcer, and R.N. Taylor, "Web-based Development of Complex Information Products," *Communications ACM*, 41(8), 84-92, August 1998.

R. Fielding and R.N. Taylor, "Principled Design of the Modern Web Architecture," *Proc. 22nd Intern. Conf. Software Engineering (ICSE 2000)*, Limerick, Ireland, 407-416, June 2000.

P.K. Garg. "Abstraction Mechanisms for Hypertext," *Communications ACM* 31(7), 862-870 1988.

P.K. Garg, P. Mi, T. Pham, W. Scacchi, and G. Thunquest, "The SMART Approach for Software Process Engineering", *Proc. 16th. Intern. Conf. Software Engineering*, 341-350, Naples, Italy, IEEE Computer Society, 1994.

P.K. Garg and W. Scacchi, "Composition of Hypertext Nodes," *Online Information 88 Proceedings*, 63-73, Dec. 1988.

P.K. Garg and W. Scacchi, "ISHYS: Designing an Intelligent Software Hypertext System," *IEEE Expert*, 4(3), 52-63, 1989.

P.K. Garg and W. Scacchi, "Hypertext System to Manage Software Life Cycle Documents", *IEEE Software*, 7(3), 90-99, 1990.

J.M. Haske and W. Wang, "Flexible Support for Business Processes: Extending Cooperative Hypermedia with Process Support," *Proc. ACM SIGGROUP Conf. Supporting Group Work*, 341-350, Phoenix, AZ, 1997.

W.L. Johnson and A. Erdem, "Interactive Explanation of Software Systems", *Automated Software Engineering*, 4(1), 53-75, January 1997.

R. Kling and W. Scacchi, "The Web of Computing: Computer Technology as Social Organization," in M. Yovits (ed.), *Advances in Computers*, 21, Academic Press, New York, 3-87, 1982.

M.R. Lowry and R.D. McCartney, *Automating Software Design*, AAAI and MIT Press, Cambridge, Mass., 1991.

T. Malone, K. Grant, K. Lai, R. Rao, and D. Rosenblitt. "Semistructured Messages are Surprisingly Useful for Computer Supported Coordination," *ACM Trans. Office Info. Systems* 6(2), 115-131, 1987.

P. Mi and W. Scacchi, "A Knowledge Base Environment for Modeling and Simulating Software Engineering Processes," *IEEE Trans. Knowledge and Data Engineering* 2(3), 283-294, 1990.

P. Mi and W. Scacchi, "Process Integration in CASE Environments", *IEEE Software*, 9(2), 45-53, March 1992.

P. Mi and W. Scacchi, "Articulation: an integrated approach to the diagnosis, replanning, and rescheduling of software process failures", *Proc. 8th. Knowledge-Based Software Engineering Conference*, 77-84, Chicago, IL, IEEE Computer Society, 1993.

K. Narayanaswamy and W. Scacchi, "Maintaining Configurations of Evolving Software Systems," *IEEE Trans. Soft. Engr.* 13(3), 324-334, 1987.

J. Noll and W. Scacchi, "Integrating Diverse Information Repositories: A Distributed Hypertext Approach," *Computer*, 24(12), 38 45, 1991.

J. Noll and W. Scacchi, "Supporting Distributed Configuration Management in Virtual Enterprises", in R. Conradi (ed.), *Software Configuration Management*, Lecture Notes in Computer Science, Vol. 1235, Springer-Verlag, New York, 142-160, 1997.

J. Noll and W. Scacchi, "Supporting Software Development in Virtual Enterprises", *Journal of Digital Information*, 1(4), February 1999.

J. Noll and W. Scacchi, "Specifying Process-Oriented Hypertext for Organizational Computing", *J. Network and Computer Applications*, 24(1), 39-61, 2001.

P.J. Nurnberg, "Extensibility in Component-Based Open Hypermedia Systems", *J. Network and Computer Applications*, 24(1), 19-38, 2001.

H. Oinas-Kukkonen, "Flexible CASE and Hypertext", *ACM Computing Surveys*, 31(4es), December 1999.

M. Porter, *Competitive Advantage*, Free Press, New York, 1980.

M.R. Quillian, "Semantic Memory," in M. Minsky (ed.), *Semantic Information Processing*, MIT Press, 1968.

T. Roth, P. Aiken, and S. Hobbs, "Hypermedia Support for Software Development: A Retrospective Assessment, *Hypermedia*, 6(3), 149-173, 1994.

E.D. Sacerdoti, *A Structure for Plans and Behavior*, Elsevier North-Holland, 1977.

W. Scacchi, "Managing Software Engineering Projects: A Social Analysis," *IEEE Trans. Soft. Engr.* 10(1), 49-59, Jan. 1984.

W. Scacchi, "On the Power of Domain-Specific Hypertext Environments," *J. Amer. Soc. Info. Science*, 40(3), 183-191, May 1989.

W. Scacchi, "A Software Infrastructure for a Distributed System Factory," *Software Engineering Journal* 6(5), 355-369, Sept. 1991.

W. Scacchi, "Understanding Software Process Redesign using Modeling, Analysis and Simulation", *Software Process --Improvement and Practice*, 5(2/3), 183-195, 2000

W. Scacchi and J. Noll, "Process-Driven Intranets: Life Cycle Support for Process Reengineering", *IEEE Internet Computing*, 1(5), 42-49, 1997.

E.J. Whitehead, "An Architectural Model for Application Integration in Open Hypermedia Environments", *Proc. Hypertext '97*, 1-12, 1997.

E.J. Whitehead, and M. Wiggins, "WebDAV: IETF Standard for Collaborative Authoring on the Web", *IEEE Internet Computing*, 2(5), 34-40, September/October, 1998.

U.K. Wiil, "HyperDisco: An Object-Oriented Hypermedia Framework for Flexible Software System Integration," *Proc. IEEE Computer Software and Applications '95*, 298-305, Dallas, TX, 1995.

U.K. Wiil, P.J. Nurnberg, and J.J. Leggett, "Hypermedia Research Directions: An Infrastructure Perspective," *ACM Computing Surveys*, 31(4es), December 1999.

W. A. Woods, "What's in a Link: Foundations for Semantic Networks," in D.G. Bobrow and A. Collins (eds.), *Representation and Understanding*, 35-72, Academic Press, New York, 1975.