The Use of Hypertext in Software Development

Susan D. Stratton
H. E. Dunsmore

Software Engineering Research Center (SERC)
Department of Computer Sciences
Purdue University
West Lafayette, Indiana 47907

SERC TR-36-P

ABSTRACT

Tools that aid in software development are called Computer Aided Software Engineering (CASE) systems or software engineering environments. These tools should lead to increased productivity and better quality software. The goal of the Software Engineering Environment Systems (SEES) project is to identify functions and capabilities that will increase the productivity of software developers but that are not available (or not implemented well) in existing software environments. We are developing and demonstrating methods for integrating such functions into existing environments as value-added extensions.

In this paper we discuss hypertext - a network of nodes of information and the links among them. Hypertext systems give users the flexibility to store information easily. This facility for storing large amounts of diverse data related in complex ways would seem to make hypertext a natural choice as a software development tool. We give a brief overview of a few hypertext-based software development systems. Some of the advantages and disadvantages of hypertext in software development are explored. Hypertext's strong points include connectivity of components, a natural user interface, and flexibility. But, there are some disadvantages including relative lack of machine processability, added complexity presented by links, node fragmentation, difficulty storing fine-grained information, and a potential of too much flexibility.

February, 1989

The Use of Hypertext in Software Development

Susan D. Stratton
H. E. Dunsmore

Software Engineering Research Center (SERC)
Department of Computer Sciences
Purdue University
West Lafayette, Indiana 47907

SERC TR-36-P

Introduction

An exact definition of what is meant by the term hypertext is really not that easy to pin down. It seems that Hypertext is becoming one of the latest "catch-words", and so there are currently several different systems available, each claiming to be The Definitive Hypertext System, and each with a slightly different view of what exactly that means. To find any agreement, it is necessary to come down to the most basic level, and simply define hypertext as "an approach to information management in which data is stored in a network of nodes connected by links".[1] The contents of nodes are meant to be discrete "chunks" of information; that is, each node is to contain a single concept or idea. Aside from this one restriction, the information in the nodes is usually relatively free format, and can contain any combination of (at least theoretically) text, graphics, photographic images, animation, sound, movies, executable code, or even more exotic things limited only by your imagination and the level of technology. These nodes are connected to each other by links, each of which serves to explicitly represent some type of relationship between the two nodes it connects. This network of nodes of information and the links among them forms the hypertext database. For a much more extensive introduction to hypertext, and a broad survey of currently available systems, see Conklin.[2]

The power of this very simple idea to a large extent rests precisely in this very simplicity. Hypertext systems give users the flexibility to store information easily in its natural structure: the complex system of various inter-relationships among components of the data can be explicitly represented by the webs of nodes and links in a hypertext database. In other words, the database can be molded to fit the data, whereas other representation schemes too often must artificially force the data to be stored to fit its own pre-defined mold. This facility for storing large amounts of diverse data related in complex ways would seem to make hypertext a natural choice for the kinds of information management problems encountered in software development.

Can hypertext really be used in software development? Is it a good choice (for a software information management scheme)? Is it the best choice? The remainder of this paper will give a brief overview of a few of the hypertext-based software development systems either currently available or in development. These systems include both "thinking tools" used to help organize information to solve complex and unstructured problems in the early stages of development, and also more CASE-like tools to deal with later, more detailed information. Finally, some of the advantages and disadvantages of hypertext will be explored in relation to the above questions.

Thinking Tools

The term "thinking tools" refers to systems which emphasize the creative, or authoring, aspects of hypertext; that is, systems intended for creating, organizing, and analyzing information on a particular subject instead of merely browsing through a previously existing hypertext database. These systems are geared to using the framework of hypertext to help a user or group of users organize thoughts and ideas toward solving some specific problem. The flexibility of hypertext, and its ability to mimic human thought processes by making wild leaps from idea to idea, make these tools ideally suited for early, unstructured thinking about complex problems. They allow users to represent disconnected ideas as they come to mind, leaving out details to be filled in later. Since software development is a complex, difficult, problem-solving process, largely unstructured at times, these "thinking tools" can be quite useful for capturing and organizing the informal part of the process.

Typical of systems which have been specifically designed for this kind of process is gIBIS, a hypertext-based tool developed by MCC supporting the IBIS method of problem solving.[3] Essentially, gIBIS supports a sort of structured argumentation among team members. Members are expected to post "Issue" nodes containing questions, such as "How should we do x?". Other team members can then respond by linking "Position" nodes to the "Issue" nodes containing possible solutions, as well as "Argument" nodes to support or refute each of the "Positions". In essence, gIBIS becomes a sort of electronic conference room, where discussions are computer-mediated and constrained to follow the hypertext organization.

Several other systems exist which can aid in the decision making process. For example, SYNVIEW is similar to gIBIS, except that team members also get to "grade" each other’s postings according to validity and relevance, in order to give weights to the various positions to help in choosing among them.[2] There are also others less-directly related. WE (or Writing Environment) is a system primarily designed to to support the up-stream part of writing when the author is collecting and organizing information. WE enforces very little structure on the network of nodes created at the beginning, but allows the user gradually to impose an increasing amount of organization as a natural hierarchy emerges.[2] EUCLID is a system which supports the construction of semi-formal reasoned arguments; argument *structure* is represented formally by links (such things as "Claim A supports claim B"), while argument *content* (that is, the actual contents of the nodes for claims A and B), is represented informally in natural language.[4] In addition, most general purpose hypertext systems can be adapted to provide some kind of problem solving support as well. For example, NoteCards has been modified to support competitive argumentation by introducing "matrix summary cards", a special node format to summarize information about a set of competing hypotheses and arguments supporting or refuting each of them.[5]

Using these tools can help analysts capture, coordinate, organize, and give at least a minimum amount of structure to the information contained in these early problem exploration design decisions which is often not captured in the formal project documentation. Perhaps just as importantly, the tools also produce a permanent record of the decisions made and the reasoning behind them for future reference, so that when the system must be redesigned, the blind alleys encountered in the first pass can be avoided. However, most of these systems are intended primarily to be "What if ..." exploration tools, and really are not suited to handle a large amount of detail. This means that at some point it is necessary to declare the issues resolved and move on to some other more CASE-like system more suitable for detailed design.

Hypertext-Based CASE Systems

Once the level of detail has moved beyond the capacity of the tools described above, hypertext can be used as the basis for CASE or CASE-like tools to continue the design process. In these systems, hypertext provides the framework in which all of the components, documentation, and other information about the project is stored.

One of these hypertext-based CASE systems is Dynamic Design, developed by Tektronix. Dynamic Design is built on an open, layered architecture with three levels. The lowest of these is the HAM, or Hypertext Abstract Machine, a generic back-end for hypertext systems.[6] The HAM is a transaction-based server providing facilities to create and manipulate nodes and links. It supports several attractive features including:

a. Attribute/value pairs which can be attached to both nodes and links, thus allowing the definition of typed nodes and links.

b. An elaborate version history scheme for nodes and links, and also for entire configurations.

c. Contexts, a partitioning scheme for clustering related nodes and links, allowing such things as organizing related nodes (such as those pertaining to a single document or module), aiding in multiple authoring without interference, and creating multiple version threads and configurations.[7]

Built on top of the HAM is Neptune, a Smalltalk based front end (user interface) designed primarily for CAD (not specifically physical design, but design in general). [8] Basically, Neptune provides a variety of graphical browsers allowing easy manipulation of objects in the HAM. Finally, on top of both of these is Dynamic Design, a CASE tool for C code development.[9] Dynamic Design holds the code, requirements, design, and documentation in a hypertext database for use by outside tools. Each component in the database is broken into "chunks" and stored in nodes. Typed links can then be created among these nodes to represent explicitly various relationships among the nodes, such as "LeadsTo", "FollowsFrom", "Implements", or "IsDefinedBy". Code analyzers exist which can automatically create an extensive number of these links in the lower levels (from the source code down), but links must be added almost completely manually for the up-stream components.

One of the primary goals of the I/SHYS project at USC is to address this problem of automating the creation of links.[10] I/SHYS is planned as an extension to DIF (or Documents Integration Facility), a software hypertext database built on top of Unix to "integrate and manage documents produced and used throughout the software lifecycle". DIF imposes a structured format on documents by allowing a super-user to define "formats" and "basic templates" for all documents, which are then filled in with the appropriate information. Links, or connections, in DIF are either captured implicitly in the hierarchy defined by the forms, or can be explicitly (manually) added between any two basic templates. DIF is simply a passive data repository; outside tools interfaced through DIF are expected to perform the actual work of manipulating and analyzing the database contents. In contrast, I-SHYS (or Intelligent Software Hypertext System) is to extend DIF into an active participant in the software process. The idea is to build in knowledge about the software development process, such as knowledge about the "agents" involved and the "tasks" they perform. Expert system assistance can then be used to automate the creation of links as much as possible, and to help guide developers through their tasks.

A final, more tenuous, example is the Leonardo project at MCC. Leonardo is a software design environment being developed to address the up-stream portion of development. One of its primary concerns is capturing (and using) all available information relevant to the design process, as well as the design itself in a central, integrated Information Base. [11] This Information Base must thus be able to deal with a large amount of very diverse forms of data coming from many different sources. As such, hypermedia is one of the technologies being investigated to represent this data. Along this line, PlaneText, a fairly simple hypertext system developed by MCC, was used as the underlying representation for the Information Base in the post - facto analysis (or vivisection) of a single large project to investigate how, and with what, to populate this Information Base.[12] PlaneText worked very well in this effort, especially after the addition of a method of simulating node and link attributes to create "views" (similar to Dynamic Design "contexts"), allowing several different kinds of information to be included easily and used in the database. The final choice for the representation of Leonardo's Information Base will probably involve at least some elements of hypertext.[13]

Can Hypertext be Used for Software Development?

Obviously it can be used, since the previous section discusses a few examples where hypertext is being used in current systems. Apparently hypertext is at least a viable scheme. In fact, one could argue that any CASE system (at least any good system) provides some degree of hypertext implicitly, even if it is not called by that name, by formalizing various relationships among the different items it stores.

Is Hypertext a Good Choice?

Using hypertext does have several strong arguments in its favor:

a) **Connectivity.** The "hidden" relationships among components are explicitly stored as links among the nodes in a hypertext database. This connectivity, knowing where things come from and what they are related to and in what ways, can be extremely important, especially when considering how changes in one component can affect the other components.

b) **Natural User Interface.** Hypertext in a way mimics the way humans think, not in a straight-line, linear progression, but in wild leaps from idea to idea (a sort of "That reminds me..." syndrome). The ability to create arbitrary links among nodes allows hypertext to follow this pattern as well, so using it as a basis leads to a very natural, intuitive human interface. In fact, the best hypertext systems require little or no training for even inexperienced users.

c) **Flexibility.** The relative lack of inherent structure in hypertext gives systems based on it a large amount of flexibility. Current CASE tools are generally very rigid about forcing users to always play by their rules; however, there are always situations that do not quite fit the mold, and hypertext tends to allow users more freedom to decide what to do in a given situation and tailor the system to fit their own needs.

Is Hypertext the Best Choice?

There are some disadvantages associated with hypertext as well:

a) **Lack of Computability.** Perhaps the greatest liability of hypertext is its relative lack of machine processability. To support the diversity required, the actual contents of nodes are usually in relatively free format, uninterpreted text and graphics. This means that virtually any interpretation of the information in the database requiring knowledge of node contents must be provided by human expertise. Machine support is limited to the level of which nodes are connected by which links, unless nodes are restricted to special classes (such as nodes containing only source code).

b) **Complexity.** Another serious problem with hypertext is the added complexity presented by the links. In any sizable project there are bound to be a tremendous number of subtle relationships among the components, to the point where everything is related to everything else somehow, and usually in several different ways. At some point explicitly documenting all these relationships with links would become such an overwhelming task that it would actually do more harm than good. In practical situations, this leaves the somewhat subjective question of deciding exactly how many (and which kinds of) links are enough.

c) **Node Fragmentation.** There is also a problem associated with the nodes. Quite often the information to be stored is so entwined that it is difficult, if not impossible, to sensibly break it up into separate nodes. In the end, these situations must usually be resolved by setting artificial cut-off points to break the information into suitably sized nodes, thus obscuring instead of clarifying the information.

d) **Fine-Grained Information.** Hypertext is really meant to store medium to large objects; trying to store very fine-grained information (such as definition-use pairs) can lead to excessively small nodes and too many links. A hybrid system somehow merging hypertext with something better equipped to store such fine-grained information (such as a relational database) may be needed to alleviate this problem.

e) **Lack of Structure.** The same flexibility and lack of structure listed as an advantage can also be a disadvantage under certain circumstances. It could be argued that hypertext may allow too much flexibility. Sometimes users "want their hands held"; they want to be told what to do and when to do it. Allowing users to make more of their own decisions also gives them more of a chance to make potentially disastrous mistakes which a more structured system would have disallowed.

Conclusions

So what then is the final answer? There are certainly some good points to hypertext and some things that it does better at than anything else available, but at the same time there are also some important areas which hypertext addresses either poorly or not at all. At the very least, there are definitely lessons to be learned from hypertext and ideas that we can use from it. Perhaps the most promising approaches being tried today involve merging hypertext with other representation schemes to compensate for the features hypertext lacks.[13, 14] The trick is to incorporate hypertext into CASE tools in such a way as to capture its greatest strengths while avoiding its weaknesses.

References

1. John B. Smith and Stephen F. Weiss, ‘‘An Overview of Hypertext,’’ *CACM*, vol. 31, no. 7, pp. 816-819, July 1988.

2. Jeff Conklin, ‘‘Hypertext: An Introduction and Survey,’’ *IEEE Computer*, vol. 20, no. 9, pp. 17-41, September 1987.

3. Michael L. Begeman and Jeff Conklin, ‘‘The Right Tool for the Job,’’ *BYTE*, vol. 13, no. 10, pp. 255-266, October 1988.

4. Paul Smolensky, Brigham Bell, Barbara Fox, Roger King, and Clayton Lewis, ‘‘Constraint-Based Hypertext for Argumentation,’’ *Hypertext ’87 Papers*, pp. 215-245, November 1987.

5. Frank G. Halasz, Thomas P. Moran, and Randall H. Trigg, ‘‘NoteCards in a Nutshell,’’ *CHI+GI 1987*, pp. 45-52, April 1987.

6. Brad Campbell and Joseph M. Goodman, ‘‘HAM: A General Purpose Hypertext Abstract Machine,’’ *CACM*, vol. 31, no. 7, pp. 856-861, July 1988.

7. Norman M. Delisle and Mayer D. Schwartz, ‘‘Contexts - A Partitioning Concept for Hypertext,’’ *ACM Trans. on Office Information Systems*, vol. 5, no. 2, pp. 168-186, April 1987.

8. Norman Delisle and Mayer Schwartz, ‘‘Neptune: A Hypertext System for CAD Applications,’’ *ACM SIGMOD Record*, vol. 15, no. 2, pp. 132-143, June 1986.

9. James Bigelow and Victor Riley, ‘‘Manipulating Source Code in Dynamic Design,’’ *Hypertext ’87 Papers*, pp. 397-408, November 1987.

10. Pankaj K. Garg and Walt Scacchi, ‘‘On Designing Intelligent Hypertext Systems for Information Management in Software Engineering,’’ *Hypertext ’87 Papers*, pp. 409-432, November 1987.

11. T. Biggerstaff, C. Ellis, F. Halasz, C. Kellogg, C. Richter, and D. Webster, ‘‘Information Management Challenges in the Software Design Process,’’ *MCC Tech. Report Number STP-039-87*, January 1987.

12. Don Peterson, ‘‘Software Design Capture,’’ *MCC Tech. Report Number STP-138-87*, May 1987.

13. Dallas E. Webster, ‘‘Mapping the Design Representation Terrain: A Survey,’’ *MCC Tech. Report Number STP-093-87*, July 1987.

14. James Bigelow, ‘‘Hypertext and CASE,’’ *IEEE Software*, vol. 5, no. 2, March 1988.