REQUIREMENTS FOR A HYPERTEXT SOFTWARE MAINTENANCE SYSTEM

Spencer Rugaber

Software Engineering Research Center
Georgia Institute of Technology

ABSTRACT

Hypertext is an appealing technology for automating the software maintenance process. Maintenance tasks such as locating an error or confirming that a modification does not break existing code are naturally implemented as link-following exploration activities in a hypertext system. This paper describes the requirements process for a hypertext software maintenance system called Hypermaint. It also discusses how the requirements affect the overall architecture of the hypertext system.

Hypertext and Software Maintenance

Software maintenance is the single most costly aspect of software development. As much as 75% of a software developer's time is spent performing maintenance activities [7]. Commercially available tools of limited functionality exist to support software maintenance. Examples are language processors, cross-referencers, and program restructurers. However, none of these tools attempt to automate the overall process of software maintenance. Moreover, they fail to take advantage of emerging technologies such as hypertext and semantic data modeling.

Hypertext is an appropriate and promising technology that addresses many of the problems confronting software maintainers. Hypertext systems are primarily intended to support exploration of "non-linear text" [3] (also called "hyperspace"). Hyperspace can be thought of as the visual manifestation of a complex information structure. Some hypertext systems are specifically designed to support a particular information structure or application arena. For example, the SAM system facilitates exploration and report generation of a single, albeit large, document that provides guidelines for designing user interface software [10]. Other hypertext systems, such as NoteCards [4], are designed to deal with general text, allowing the user or system administrator to specify "links" (explicitly designated, possibly attributed connections) among text "nodes" (sections of text).

The purpose of this paper is to describe the requirements that a hypertext system must satisfy in order to address the software maintenance problem. In particular, it discusses two topics: 1) what is the maintenance process, what activities comprise it, and how can they be facilitated by a hypertext system? 2) Of the many architectural possibilities for a hypertext system, which choices are best suited to support maintenance? These questions will be answered as part of the specification and design steps of the development of Hypermaint, a hypertext software maintenance system.

Modeling Software Maintenance

Software maintenance consists of those activities which involve the modification of software while keeping its major purpose intact [1]. It is the single most costly part of the software development process and has been largely ignored by the research community. Maintenance has been characterized as being either corrective, adaptive, or perfective depending on whether its intent is to correct problems, to adapt to new environments, or to add features. In all cases, the source code comprising the software must be examined to ascertain its overall purpose and function and to determine the area in which a specific modification will be made. It must be understood so that the appropriate alterations may be made and so that all consequences of the modifications on the remainder of the program can be detected and investigated.

A model of software maintenance must be consistent with observations and experimental results from several other areas of computer science. In particular, we are looking at the following five fields: 1) debugging - how do programmers explore and understand code in order to determine the location of an error? 2) software psychology [12] - which (human) factors bear upon the ease of comprehending and modifying code? 3) error data studies - what types of errors are made by programmers and in which programming language constructs do they occur? 4) cognitive science [13] - what models of the program understanding process are relevant to maintenance tasks? 5) software engineering [8] - how must the source code intensive aspects of software maintenance fit in with the overall software engineering process.

An example of a requirement concerns indentation. The use of indentation can improve the comprehensibility of a program. The exact amount of indentation to use, however, is questionable. This leads to a possible requirement to support user-specified indentation levels for displayed programs. That is, the user of Hypermaint would be able to control how a program is displayed, independently of how it was originally written.

Another example concerns mistake-prone programming language constructs. For a particular language, certain constructs, such as the looping mechanism, will be the locus of a disproportionally large number of programmer errors. This indicates that a hypertext system to support error finding should be able to make the maintenance programmer aware of uses of these constructs.

Results from these five areas together comprise a set of requirements for Hypermaint. They must, however, be integrated into an overall model of the process that can then be automated. The modeling will be done using semantic data modeling [6, 9]. This approach has two benefits. First, it serves as a formalism for expressing the model. Because of its generality, it is well-suited for dealing with requirements from disparate domains. Secondly, the resulting data model can serve as a description of the information structure that underlies the hypertext system.

Software Maintenance Activities

The software maintenance process involves several activities that should each be supported by tools. (The following discussion is couched in terms of error correction maintenance tasks but could as well address enhancement and adaptation.) A program to be modified first needs to be explored in order to gain an understanding of its overall function and the possible location of a problem. The preliminary exploration leads to the construction of a hypothesis about the nature of a problem and how it should be fixed. The hypothesis must then be confirmed. This involves searching for possible side effects of a modification, including running experiments. The actual modification, retry, and testing processes comprise the next activity. Finally, overall management is needed in order to facilitate coordination and thoroughness.

Together, all these activities form the components of a maintenance process model. Requirements from the five areas described in the previous section serve to put constraints on how the activities should be carried out and in what form the intermediate results should be represented.

Global Requirements

There are, of course, other requirements that constrain a hypertext software maintenance system. Of particular interest are the questions of populating the hypertext information structure, the user interface presented to the maintenance programmer, and the integration of the system with other software tools, either existing or built upon the hypertext system.

One of the most difficult problems in building hypertext systems is automating the process of building the underlying information structure. In the case of source code maintenance, it should be possible to use the techniques of optimizing compilers, particularly data flow analysis, to automate this task. For example, the determination of the set of statements that could affect the value of a variable contained in a node [14] is an example of the "Reaching Definitions" data flow analysis problem [5].

Hypertext systems support the process of exploring non-linear text. The exploration may involve link following, searching, or browsing guided by a graphical representation of the information structure. The presentation of the data and the interaction required to explore it are part of the user interface to the system. So too are the activation mechanisms for the tools that support the maintenance activities. It is intended that this interface takes advantage of large screen, bit-mapped displays, and "point-and-click" operation invocation.

Powerful systems are realized when an extensible set of primitive operations can be combined by a small set of "combining forms". A popular example is the "Bourne" shell interface to the UNIX operating system* [2]. It is desirable that Hypermaint be constructed in such a way that new tools can be easily added by the maintenance programmer. It is also desirable that the system allow easy integration of existing tools, such as pretty printers or cross referencers.

The Architecture of Hypertext Systems

Conklin, in his impressive survey of hypertext systems [3], presents a table listing features of current hypertext systems. An example feature is whether the links between nodes have data types. Typed links serve as a further organizing discipline for the set of nodes. Some hypertext systems provide typed links, and others do not. This is an example of an architectural design decision that confronts the developer of a hypertext system. For Hypermaint, the decisions will be based on the requirements described above. In the case of maintenance, typed links can serve to implement relations taken from the semantic model. In this way, the maintenance process model will drive the design process for Hypermaint.

Another design issue from Conklin's list concerns version control. Of course, this is an important software engineering concern, and exploration of old versions of the source code should be facilitated by Hypermaint. What is not clear, however, is who should be responsible for providing the versions. Possibilities include the file system, the data base management system, or including old versions as nodes in the information structure itself.

One other example relates to searching for strings. In software, a given collection of text characters may be used in a variety of ways: as the name of a variable, within a comment, or as part of literal string. Hypermaint's search facility should be aware of these differences and provide the user with a mechanism for searching for the correct type of string occurrence.

Conklin's table serves as a list of issues to drive Hypermaint's design process. The resolution of the issues depends on the results of the requirements process described above.

Summary

This paper has described the requirements process that will lead to a functional specification of a hypertext software maintenance system called Hypermaint. When this is combined with Conklin's list of features, an overall architectural design can be constructed. The actual implementation will include the use of user interface software (built on the X Windows System [11]), a heavily instrumented compiler (in which data flow and symbol table information is exported), and an underlying database system to support the information structure.

Of course, specific tools must also be built to automate the maintenance activities discussed above. Initially, the exploration activity is targeted and two tools will be built. They are a graphical browser for viewing the overall program structure and a "grazer" for making detailed examinations of specific program constructs.

* UNIX is a trademark of AT&T Bell Laboratories.

Acknowledgements

I would like to thank Timothy Tiemens and Victoria Tisdale for conversations that helped formulate the ideas presented in this paper.

References

[1] Barry W. Boehm, Software Engineering, *IEEE Transactions on Computers*, Vol. C-25, No. 12, December 1976.

[2] S. R. Bourne, An Introduction to the UNIX Shell, In *UNIX User's Manual / Supplementary Documentation*, University of California, 1980.

[3] J. Conklin, Hypertext: An Introduction and Survey, *IEEE Computer*, Vol. 20, No. 9, September, 1987.

[4] Frank Halasz, Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems, *Communications of the ACM*, Vol. 31, No. 7, July 1988.

[5] Matthew S. Hecht, *Flow Analysis of Computer Programs*, North-Holland, 1977.

[6] Richard Hull and Roger King, Semantic Database Modeling: Survey, Applications and Research Issues, *ACM Computing Surveys*, Vol. 19, No. 3, September 1987.

[7] Bennet P. Lientz, Issues in Software Maintenance, *ACM Computing Surveys*, Vol. 15, No. 3, September 1983.

[8] Jay Arthur Lowell, *Software Evolution*, John Wiley, 1988.

[9] Joan Peckham and Fred Maryanski, Semantic Data Models, *ACM Computing Surveys*, Vol. 20, No. 3, September 1988.

[10] Gary Perlman, Hypertext: An Information Delivery Mechanism, Talk given at Georgia Institute of Technology, March 29, 1988.

[11] Robert W. Scheifler and Jim Gettys, The X Window System, *ACM Transactions on Graphics*, Vol. 5, No. 2, April 1986.

[12] Ben Shneiderman, *Software Psychology: Human Factors in Computer and Information Systems*, Winthrop Publishers, 1980.

[13] Eliot Soloway, Jeffrey Bonar, and Kate Ehrlich, Cognitive Strategies and Looping Constructs: An Empirical Study, *Communications of the ACM*, Vol. 26, No. 11, November 1983.

[14] Mark Weiser, Program Slicing, *5th International Conference on Software Engineering*, IEEE Computer Society, March 1981.