

Incremental Redocumentation with Hypertext

Vaclav Rajlich
Department of Computer Science
Wayne State University
Detroit, MI 48202
rajlich@cs.wayne.edu

Abstract

Redocumentation is the recovery and recording of software comprehension. Since software comprehension is the most expensive part of software maintenance, redocumentation is the key to software maintainability. This paper describes the process and the tools of incremental redocumentation where the comprehension of the software is recorded in hypertext, in the style of the World Wide Web. The paper describes the tools which support redocumentation, and gives several examples.

1. Introduction

Software redocumentation is part of software reengineering. While reengineering may involve additional activities like restructuring the code, retargeting, etc., redocumentation only recovers the understanding of the software and records it, and therefore makes future program comprehension easier. Program comprehension is the most expensive part of software maintenance, and therefore program redocumentation is the key to software maintainability. Redocumentation can either be done in a whole-scale effort, or it can be done incrementally as a part of the maintenance miniprocess. In the second case, the understanding gained in each maintenance visit is recorded, and the new documentation is accumulated. Since the introduction of the World Wide Web, hypertext has become a very popular and familiar technology. The basic tools such as the html language, browsers, Java applets, etc. are widely available and understood by many programmers. The same technology can be used in software redocumentation. The hypertext in which documentation can be accumulated is practically unlimited, ensuring that there is no need to be overly restrictive over the issue of what will be

recorded. The key to the successful use of this technology is the structure of the documentation, which must support easy access to the information which the programmer needs and trusts.

The structuring of the hypertext documentation we describe here is based on cognitive theories of program comprehension. The information is divided into annotations of individual program constructs (packages, classes, functions, etc.). For each construct, there are different partitions of the documentation, each recording a different level of abstraction or different viewpoint. The code and documentation partitions are browsed by standard hypertext browsers known from the World-Wide Web, such as Netscape, Mosaic, etc.

In the next section, we describe the principles of the partitioned annotations of software. Section 3 contains examples. Section 4 describes the incremental redocumentation process by which the annotations are accumulated. The tools supporting the documentation are described in section 5. Section 6 contains the conclusions. The Appendix contains an example of an annotation skeleton.

2. Partitioned annotations of software (PAS).

The field of software comprehension is summarized in [3,10]. It is an interdisciplinary field, spanning both software engineering and cognitive science. A number of strategies describing how programmers understand code have been investigated. One strategy (bottom-up) is based on so-called chunking. Chunks are pieces of code which have an identifiable meaning, and are recognized by the programmer. Chunks in turn can be parts of bigger chunks. The programmer who is using chunking as his comprehension strategy first recognizes the smallest chunks, then the bigger ones, etc., until enough of the program is understood.

The top-down strategy of comprehension is based on hypotheses about the software and their verification. The programmer using this strategy first creates a set of hypotheses which may depend on each other, and then looks for evidence (beacons). If beacons are found, hypotheses are confirmed and the program is understood. If they are not confirmed, they are discarded and another set of hypotheses is adopted.

An interesting variant of the top-down strategy was presented in [14]. In it, the programmer searches the program for concepts known from the program domain and tries to determine where these concepts are implemented. If, for example, the program domain is the domain of debuggers, the programmer searches for breakpoints and other concepts from that domain.

The partitioned annotations of software (PAS) serve as a notebook of the maintenance programmer, in which the programmer can record all of his understanding, whether it was arrived at in a top-down or bottom-up fashion, whether it is complete or partial, or even whether it is a confirmed or tentative. Since the PAS is in hypertext in the style of World Wide Web, there is no need to limit the number of partitions or their contents, and a partition in which it can be recorded can always be found.

From the global point of view, the PAS annotations are a matrix, where one coordinate is the constructs of the code, and the other coordinate is the selected partitions. There is a wide variety of partitions which a programmer may want to use, with various levels of abstraction and various kinds of information. Hence, each entry in the matrix is the annotation for a specific construct, containing information of a specific kind. Among the partitions, domain partitions play a special role.

Each program operates in a certain domain of application, and each application domain has an ontology, consisting of essences and concepts of that domain. They have to be supported by the program, and they are important for the understanding the program. For example, if the application domain is that of library, then the concepts supported are books, loans, customers, etc. These concepts in turn have to be supported by specific constructs in the code. A domain partition makes these constructs comprehensible. It is written in terms of the domain concepts only, and is understandable to both a programmer and to a user of the program.

3. Examples.

In object oriented languages, the backbone of the system is classes and their dependencies. Let us consider a library system which registers book loans in a small library. The system consists of several classes. The annotations of classes ConMenu and LoanList are:

Domain annotation of ConMenu

This class provides the user interface for the whole application. It provides functions which display the menu of options (check out book, return book, display list of books taken out by a particular customer, display all the customers, display all the books), read the user response, and execute the corresponding option.

Representation annotation

Integer "choice" stores the choice opted by the user.
"custlist" contains the list of the library customers.
"booklist" contains the list of all books.
"loanlist" contains the list of all current loans.

Domain annotation of LoanList

This class maintains a table of the bar codes of the books checked out, together with card numbers of the customers. It provides functions to check out and return books. It also provides a function to display the list of books checked out by a particular customer, as well as his details.

Representation annotation

"head" contains the address of the list of loans. It is a pointer to the class LoanBook.

In functional programming in the language C, the backbone of the system is the functions and the call relationship between them. The most important annotations are again the domain annotations, which explain the meaning of the function in terms of the domain, and "algorithm" annotations, which explain which functions are called and how. For example, the following are the domain and algorithm annotations of the function Wm_ActiveBrowser of tool VIFOR [2]:

Domain Layer for Wm_ActiveBrowser

This function sets or returns the window-id of the active browser. An active browser is the one to which subsequent operations will apply.

Algorithm Layer

```
If action is 1 then
  If wid is Panel or Menu window then
    Return 0
  If wid is not active browser then
    Make wid as active browser.
  Return 1.
If action is 0 then
  Return active browser window-id.
```

4. Incremental redocumentation process

Incremental redocumentation is done as a part of the maintenance miniprocess. The understanding gained in each maintenance visit is recorded in hypertext, and the resulting documentation is accumulated over time. The miniprocess of software change then has the following form:

- Request for change
- Understanding the current system
- Localization of the change
- Implementation of the change
- Ripple effects of the change
- Verification
- Redocumentation*

During the early phases of the miniprocess, the programmer gains an understanding of a particular aspect of the system. He uses that understanding to execute the change. Finally, at the end of the miniprocess, he records it so that it is not forgotten. As the software is repeatedly changed, and as additional constructs are visited, more and more documentation is accumulated.

Whenever incremental redocumentation is employed, there is no need for up-front investment in redocumentation. The redocumentation is done as part of the maintenance minicycle; and it is based on the understanding which the maintenance programmer acquired during the process of change. Hence, compared to whole-scale redocumentation, it is relatively inexpensive. The accumulated documentation is concentrated in the parts most often visited, which are also the parts where the documentation is most needed. Hence, the incremental redocumentation naturally directs the attention of the redocumentation effort to the parts with highest payoff.

The quality of the annotations can be verified by walk-throughs, where other programmers comment on their accuracy, usefulness, readability, and style. Since

annotations are easily changed, as their change does not require recompilation, they can be edited and improved during the walk-throughs.

5. PAS tools.

PAS tools belong to three separate categories: PAS browsers, PAS editors, and PAS generators. They can be combined into one tool, or they can be implemented separately.

PAS browsers are the tools which support browsing through PAS documentation. The existing World Wide Web browsers like Mosaic, Netscape, Explorer, etc. deal with this task satisfactorily, and therefore there is no urgent need to develop specific PAS browsing tools. PAS editors support editing and updating PAS documentation. Again, the existing universal editors or specialized World Wide Web editors, such as html editors, deal with this task satisfactorily. However, PAS generators are specific for PAS documentation. They parse existing code and prepare the skeleton files for PAS documentation. In this way, they make the starting phase of PAS easier, and guarantee that PAS will have a uniform structure and appearance across the whole system. Universal PAS tools combine the functions of generation, editing, and browsing in one tool.

This section describes two examples of PAS tools: VIFOR 2 [2], which is a universal PAS tool for the functional programs written in languages C and Fortran, and HMS, which is a PAS generator for programs written in object oriented language C++.

VIFOR 2 is a tool which combines code browsing capabilities with PAS documentation. The code browsing part is identical to the original VIFOR 1 [4], which supports the development and maintenance of C and FORTRAN programs. The user interacts with the code browser through a display of programs in two different forms: the traditional code, or the graphic representation. VIFOR contains transformation tools for both directions i.e., from code to graphics and from graphics to skeletons of code. A parser extracts from the code all information about declarations and their relationships. This information is stored in a database. The database contains information on the global declarations, which include both functions and global variables, and user defined data types. It also contains information about their relations, i.e. calls between the functions, reference between functions and global variables, relationship between global variables and user defined types. This data model is modified for FORTRAN, where there are no user defined data types, and COMMON constructs replaces global variables.

The user enters his requests through the Menu window and Query panel window. The Query panel window specifies the type of query the user wants to make. The Menu window controls the whole application, and supports user requests such as "Open a new project", "Edit program", etc.

VIFOR 2 is a VIFOR 1 code browser extended with a Mosaic hypertext browser. This means that VIFOR 2 directly operates Mosaic, by specifying for it the documents to view, directing it to open new windows, etc. The VIFOR 2 interface consists of the old interface of VIFOR 1, augmented by a new menu panel called "Document menu". The document menu has three items: "Edit Document", "View Document" and "Add Partition".

HMS is a PAS generator for programs written in object oriented language C++. Its input is a directory of C++ source files and header files. The tool parses all files in the directory and creates a skeleton documentation file for each class. Skeleton documentation files are written in html and contain all information extractable from the code. They also contain additional empty spaces for any documentation the programmer may want to add. For each partition, there is section of the file with a header and a space for the documentation comments. Appendix contains an example of a skeleton file.

The skeleton contains domain annotations for the class and for all its function members. Since domain annotation cannot be extracted from the code, the skeleton file contains only a header and an empty space for future annotation. Similarly, algorithm annotations for member functions contain only a header and an empty space. In contrast, the annotations for the class representation and for the representation of the member function arguments are partially extracted from the code. The graph of dependencies and the member function list are both fully extracted from the code. All links to the code of the class and the definition of the class are also both fully extracted. The user can add additional partitions when running HMS. The tool queries the user for the desired partitions and creates the appropriate header and space for them in all documentation files.

PAS generated by HMS is edited by standard editors, and browsed by the World Wide Web browsers like Mosaic, Netscape, Explorer, etc.

6. Conclusions.

In our experience, the partitioned annotations of software (PAS) proved to be a superior way to document

software [6]. In this paper, we described the process of incremental redocumentation based on PAS, as well as two tools supporting PAS.

PAS has been used in several settings: as part of student projects in software engineering classes, as part of an ongoing research project, and in industrial projects. In each of these situations, it substantially improved the maintainability of the software. Based on that, we expect PAS to find an application in a wide variety of software maintenance projects.

References.

- [1] R. Brooks, "Towards a theory of the comprehension of computer programs." *International Journal of Man-Machine Studies* 18, No.6 (1983) 543-554.
- [2] Rajlich, V., Adnapaly, S.R., VIFOR 2: A Tool for Browsing and Documentation, Proc. 1996 IEEE International Conf. on Software Maintenance., 296-300
- [3] T. A. Corbi, Program Understanding: Challenge for the 1990s, *IBM Systems Journal*, Vol 28.
- [4] V. Rajlich, N. Damaskinos, W. Khorshid, P. Linos, VIFOR: A Tool for Software Maintenance, *Software Practice and Experience*, Jan, 1990, 67-77.
- [5] E.J.Younger and K.H.Bennett, Model-Based tools to Record Program Understanding. In: Proc. 1993 IEEE Workshop on Program Comprehension, 87-95.
- [6] V. Rajlich, R. Gudla, J. Doran, Layered Explanations of Software: A Methodology for Program Comprehension, In: 1994 IEEE Workshop on Program Comprehension, 46-53.
- [7] Bigelow, J. "Hypertext and CASE." *IEEE Software*, Mar.1988: 23-27
- [8] P. Brown, "Integrated Hypertext and Program Understanding Tools," *IBM Systems Journal*, Vol. 30, No. 3, 1991
- [9] W. Horton, "Is hypertext the best way to document your product?: An Essay for Designers," *Technical Communication*, 38:20-32 (Feb' 1991).
- [10] Robson, D.J., Bennett, K.H., Cornelius, B.J., and Munro, M. "Approaches to Program Comprehension." *Journal of Systems and Software* 14 (1991), 79-84.
- [11] N. Fletton, M. Munroe, Redocumenting software systems using Hypertext Technologies, Proc. CSM 1988, 54-59
- [12] Horowitz, E. and Williamson, R.C., "SODOS: A Software Documentation Support Environment-- It's use." *IEEE Transactions on Software Engineering* 12 (11) (1986) 1076-1087.
- [13] von Mayrhauser, A., Vans, A.M., On the Role of Hypotheses During Opportunistic Understanding While Porting Large Scale Software, Proc. 4th IEEE Workshop on Program Comprehension, 68-77.
- [14] T.J. Biggerstaff, B.G. Mitbander, D.E. Webster, Program Understanding and Concept Assignment Problem, *Communications of ACM*, May 1994, 72-83.

Appendix: Skeleton annotation from HMS

Class CAnnotDlg

- Definition of the class
- Domain annotation
- Representation annotation
- Member functions
- Code of the class
- Dependencies of the class

Domain annotation

//Domain annotation of the class goes here.

Representation annotation

- m_CustomClassAnnotList is of the type CStringList

Dependencies for CAnnotDlg

```
+-----+
| CAnnotDlg |
+-----+
|
|   +-----+
|----| CClassInfo |
|   +-----+
|
|
|   +-----+
|----| CFunctionObject |
|   +-----+
|
```

Member functions of the class

- BOOL OnInitDialog()
- void OnOK()

BOOL OnInitDialog()

- Domain annotation
- Algorithm
- Representation annotation for each argument

Domain annotation

//Domain annotation of the function goes here.

Algorithm

//Algorithm of the function goes here.

Representation annotation for each argument.

None.

void OnOK()

- Domain annotation
- Algorithm
- Representation annotation for each argument

Domain annotation

//Domain annotation of the function goes here.

Algorithm

//Algorithm of the function goes here.

Representation annotation for each argument.

None.