

Incremental Redocumentation Using the Web

Too often, a company takes over someone else's software only to find that documentation is sadly lacking. This report shows how one small software company used the Web-based Partitioned Annotations to cost-effectively evolve a software application it had inherited.

Václav Rajlich, Wayne State University

Partitioned Annotations of Software is a Web-based medium that records a programmer's comprehension of a particular program and makes it available to the programmer's teammates. PAS divides annotations for individual software components into specialized partitions, so programmers can easily find the specific information they need. This article reports on one small company's experience in maintaining a Window-based application that was initially developed elsewhere.

The company's project team created PAS incrementally during regular maintenance, thereby greatly diminishing problems associated with code ownership, such as programmer scheduling and the loss of understanding during personnel turnover. As I'll show, PAS thus contributed significantly to the company's ability to evolve the application.

The Problem They Got

The program involved was a shrink-wrap software package for point-of-sale use, with over one million lines of C++ code. Originally written in the US, the software's maintenance and evolution moved offshore to India in 1997 where a new team of programmers was entrusted with maintaining it. Some of the original people remained involved and provided continuity.

From the outset, the project team faced several fundamental issues. Because it was written under strong competitive pressure

and no significant documentation was produced, the original code arrived with very little documentation. The new team could have made "scorched-earth" changes in the form of patches and wrappers, but those shortcuts rapidly destroy a program's coherence and make an application unmaintainable. Early on, the team rejected that route and decided to invest instead in program comprehension, so that the maintenance and evolution changes would respect—even enhance—the program's architecture.

Program comprehension is a precious commodity, consuming more than half of a software maintenance and evolution work.¹ Often, the comprehension is not recorded and resides entirely in the programming team. Because no single programmer can entirely understand large applications like the point-of-sale project, managers divide them into parts, assigning each part to a different programmer. Because a small project team can ill afford re-

dundancy, each part is comprehended, or “owned,” by one specific programmer.²

In that situation, personnel turnover poses a significant threat. A key programmer’s resignation can have serious consequences because the comprehension—effectively half the work he or she has done—departs as well. The project team then must rebuild that part’s comprehension from scratch, often at a considerable expense.

Code ownership also impairs a team’s ability to schedule programmers to maintenance and evolution tasks. Because change requests are unevenly distributed, teams having code ownership will overwork some programmers while leaving others underutilized. In a small team, maintenance management will suffer, wasting resources and draining profits.

To address these issues, the company adopted an incremental redocumentation strategy. Incremental redocumentation rebuilds the documentation incrementally and opportunistically. Whenever a programmer makes a change and obtains a comprehension, that comprehension is recorded in a shared medium, accessible to every team member. As I’ll describe, the company used PAS as the medium. These are Web-based hypertext annotations that can be browsed by standard Web browsers such as Netscape or Explorer.

So What IS PAS?

PAS was inspired by theories of program comprehension.^{3,4} (See the “Related Work” sidebar for a discussion of PAS’s role in relation to similar efforts.) The PAS medium is a hypertext notebook in which a programmer can record software understanding and various observations about software.⁵ Each code component—each class, function, dependency, or function argument—has its own annotation that explains its meaning. This annotation divides into several partitions, each describing the component at a different level of abstraction or from a different viewpoint. All project classes share the same partitions, as do all functions, dependencies, and function arguments. Embedding among the code’s components is reflected by embedding among the annotations, so member function annotations are embedded in the class annotations, and function argument annotations are embedded within the function annotations.

The annotations are partitioned because a user usually needs only a small fraction of

The business case for incremental reengineering argues that an incremental opportunistic process is less risky and easier to justify than a whole-scale process that reengineers the whole application at one time. Incremental redocumentation described in this article is a special case of the incremental reengineering Michael Olsem described, and the same business case applies here.¹

The PAS differ from other attempts to use hypertext annotations of programs, because PAS are partitioned. Annotations for each software component consist of several partitions, each describing the component from a different viewpoint. In contrast, monolithic annotations²⁻⁵ provide one large annotation for each component, so the programmer must search for the desired information within it. This search complicates the process of program comprehension. Because monolithic annotations will be repeatedly read in their entirety, programmers are reluctant to enter unconfirmed or subjective information. However, in some situations, this might be the only available information, so its loss lessens the maintainability of software.

PAS was developed within the context of program comprehension research, showing promise in an early experiment.⁶ First implementation of PAS was a part of a browser we developed.⁷

Hypercode offers a different approach to using the Web for software documentation.⁸ In hypercode, the code and documentation are cross-referenced by hypertext links, which also point from code to documentation and between the components of the code. Compared to PAS, hypercode provides greater support for the browsing, at the expense of more difficult updates when a change is made.

The generator of annotations HMS uses program analysis that extracts software components and their dependencies. There has been a substantial amount of work done in program analysis. The closest work deals with the browsers that extract components and dependencies from the code, store them in a database, and let the programmer query the database. They allow browsing from one component to the neighboring ones.^{9,10} Parsing techniques for legacy code and hypertext-based redocumentation in a large company has also been studied.¹¹

References

1. M.R. Olsem, “An Incremental Approach to Software Systems Re-engineering,” *Software Maintenance: Research and Practice*, Vol. 10, No. 3, May/June 1998, pp. 181–202.
2. J. Bigelow, “Hypertext and CASE,” *IEEE Software*, Vol. 7, No. 2, Mar. 1988, pp. 23–27.
3. N. Fletton and M. Munroe, “Redocumenting Software Systems Using Hypertext Technologies,” *Proc. IEEE Conf. Software Maintenance*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1988, pp. 54–59.
4. E. Horowitz and R.C. Williamson, “SODOS: A Software Documentation Support Environment,” *IEEE Trans. Software Engineering*, Vol. 12, No. 11, 1986, pp. 1076–1087.
5. E.J. Younger and K.H. Bennett, “Model-Based Tools to Record Program Understanding,” *Proc. IEEE Workshop Program Comprehension*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1993, pp. 87–95.
6. V. Rajlich, R. Gudla, and J. Doran, “Layered Explanations of Software: A Methodology for Program Comprehension,” *Proc. IEEE Workshop Program Comprehension*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1994, pp. 46–52.
7. V. Rajlich and S.R. Adnapaly, “VIFOR 2: A Tool for Browsing and Documentation,” *Proc. IEEE Int’l Conf. Software Maintenance*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1996 296–300.
8. G.E. Kaiser et al., “An Architecture for WWW-Based Hypercode Environments,” *Proc. Int’l Conf. Software Engineering*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1997, pp. 3–13.
9. H.A. Muller and K. Klashinsky, “Rigi—A System for Programming-in-the-Large,” *Proc. Int’l Conf. Software Engineering*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1988, pp. 80–86.
10. Rajlich, V., N. Damaskinos, W. Khorshid, P. Linos, J. Silva, Visual Support for Programming-in-the-large, *Proc. IEEE Conf. Software Maintenance*, IEEE CS Press, 1988, 92–99.
11. A. van Deursen and T. Kuipers, “Building Documentation Generators,” *Proc. Int’l Conf. Software Maintenance*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1999, pp. 40–49.

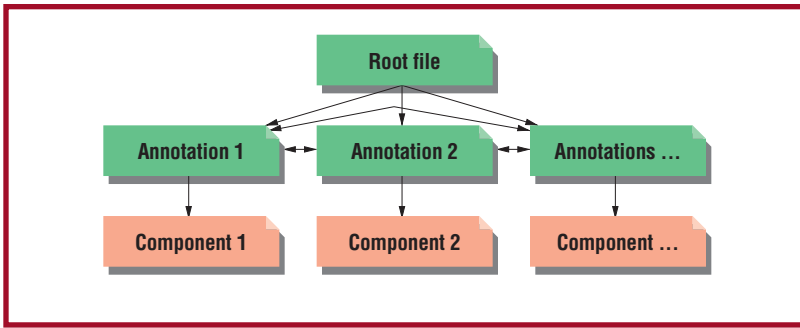


Figure 1. The PAS architecture.

the voluminous documentation.⁶ PAS lets the user find and read only relevant partitions while skipping the rest. It also lets the documentation's author add new information to the right partition when it is available. Figure 1 shows the PAS architecture.

The entry point to the PAS is the root file, which contains a list of all classes and hyper-text links to their annotations. The class annotations have pointers to the class code. When browsing through the PAS, the user moves between the code and the annotations, using both "forward" and "backward" directions. Because the code is a leaf in the tree of annotations, it does not have any embedded hypertext links. Consequently, it needs no change when annotations change or move to different files or directories. The code and annotation updates are independent of each other, making independent updates of annotations practical.

The partitions used for classes in the point-of-sale project are "domain annotation," "class dependencies," "dependency annotation," "author's comments," and "member functions." The domain partition explains the classes in terms of the application domain, thereby making these classes comprehensible on this abstract level to everybody who understands the domain. In the point-of-sale program, the domain partition contains terms "credit card," "expiration date," "credit card number," and so forth. Reading domain partition is usually the first step to the comprehension of the code and the first step in the change, because change requests are formulated in domain terminology.

The dependency partition is another important partition. Program classes depend on each other, and the dependencies are found by code analysis. However, programmers often must know what the dependencies mean—they need to know what particular role the supporting classes play or what particular service the dependent classes require.

PAS can contain additional partitions, based on the specific project needs. In the point-of-sale project, an extra partition

called "author's comments" contains all other relevant information that does not fit into previous partitions. Class annotation also contains the list of member functions. Each member function has several partitions of its own, as do function arguments.

Incremental Redocumentation Process and Tool

The change request is the first step in the change process. It is in plain English and usually originates with the customers, passing to the project team through the application's owners and distributors.

During the planning phase, the team identifies the parts of the software affected by the change and assigns a particular programmer to implement the change. The programmer then implements the change and verifies its correctness. Finally, during the redocumentation phase, the program comprehension gained during the change is recorded in the appropriate partitions of PAS.

As they plan and implement the change, programmers use standard Web browsers such as Mosaic, Netscape, or Explorer to read the PAS. Partitioning lets them read only the relevant parts of PAS.

The annotations are created after the change has completed, during the redocumentation phase. PAS are partially extracted from code and partially produced by a manual process. To guarantee uniformity and automate the mechanical tasks, the team used a specialized extraction tool, called Hypertext Management System. Prototype HMS was developed at Wayne State University and then improved and maintained by the team.

HMS distinguishes between mandatory partitions that are always generated and elective partitions that are generated on a user's demand. Mandatory partitions are

- domain and dependencies partitions for the classes,
- domain and algorithm partitions for member functions, and
- domain partition for arguments.

Elective partitions vary from project to project and are based on the project needs. The point-of-sale project used the elective partition "author's comments" where all additional useful information was recorded.

When creating new PAS skeletons for

Figure 2. Example of PAS annotation generated by HMS.

previously undocumented code, the user supplies the following information:

- directory of a C++ program to be annotated,
- class members to be annotated—for example, public and protected functions,
- elective partitions, and
- the PAS directory.

Based on this information, HMS parses the program files and creates skeleton PAS files in HTML format. The files contain blank spaces for any documentation the programmer might want to insert (see the example in Figure 2). In Figure 2, all red highlighted information was entered manually by the programmers, the rest is generated automatically by HMS. The team's programmers used a standard editor to manually complete the PAS skeletons.

Meticulous adherence to coding conventions made the task of parsing the code much easier. Each class has two files: a header file that also contains all `#include` statements and a `.cpp` file that contains the bodies of the function members. The HMS parser relies on this convention.

During a change, classes can be deleted or new classes can be introduced, and the same is true for class members, dependencies, and function arguments. Because PAS reflects the software's structure, it changes as well. To preserve the relevant and unchanged parts of the old PAS, HMS parses both the code and old annotations. It deletes from PAS all annotations for the classes, class members, dependencies, and function arguments that disappeared in the change, adding new annotation skeletons for classes, class members, dependencies, and arguments that were newly introduced. (Because of overloading, adding or deleting an argument to a function can complicate the search for the ancestor function in the old program. In a rare case of ambiguity, HMS might identify a wrong function as the ancestor.) The next manual step will fill in newly generated skeletons with the fresh information, and correct obsolete information in the partitions affected by the change.

HMS also supports addition or deletion of elective partitions. If a specific elective partition is deemed obsolete, it can be removed from all PAS files. Similarly, HMS allows ad-

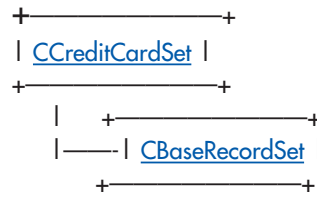
Class CCreditCardSet

- [Definition](#)
- [Code of the class](#)
- [Domain annotation](#)
- [Dependencies](#)
- [Dependency annotation](#)
- [Author's comments](#)
- [Member functions](#)

Domain annotation

This class accesses the CREDIT_CARD table to fetch a list of all the credit cards accepted by the system. This class is also used to search, add, modify and delete records.

Dependencies



Dependency annotation

CBaseRecordSet

This class is the base class for all the classes that accesses tables from the database. This class provides functions to add, modify, delete, search and fetch records from a table.

Author's comments

All the database access code is in CBaseRecordSet.

Member functions

- void [InitFields\(\)](#)
- CString [GetCode\(\)](#)
- CString [GetName\(\)](#)
- CString [GetAuthorization\(\)](#)
- void [SetCode\(CString strCardCode\)](#)
- void [SetName\(CString strCardName\)](#)

...

void SetCode(CString strCardCode)

- [Domain annotation](#)
- [Algorithm](#)
- [Annotation for arguments](#)

Domain annotation

This function sets the credit card code before adding or modifying a record.

Algorithm

Set the argument value to the field member variable.

Annotation for arguments

- strCardCode: This argument is set to the card code variable.

...

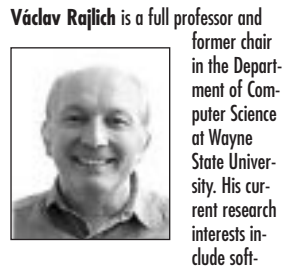
dition of new elective partitions whenever new needs arise in the project. For that, HMS also parses the old PAS and adds skeletons of the new partition to all PAS files.

The company's managers realized very early that the redocumentation task would not be successful if done by the maintenance and evolution programmers themselves. Annotating the program and maintaining it require two different skill sets. Also, because the redocumentation entails an immediate expense of time and energy but has a deferred benefit, it requires a high level of citizenship or substantial pressure from company management. However, the top priority for the maintenance and evolution programmers is to make the correct changes to the software. Consequently, because these programmers have a different and more important project responsibility, their managers are reluctant to press them to do the redocumentation as well.

For that reason, the point-of-sale team employed trainee programmers whose only job was to create, maintain, and fine-tune the PAS documents. They based their work on informal release notes written by the maintenance programmers that described the change. The trainee programmers started with this document, read the source code, and wrote the documentation. They were expected to write understandable annotations, interviewing the maintenance programmer to resolve ambiguity or uncertainty. Because the annotations were their main contribution to the project and they were evaluated on that basis, they invested their best effort in the PAS. This approach is analogous to the widespread practice of using specialized writers to develop user manuals and user help systems. The writers' expertise centers on communication with the human users, expertise that PAS also requires.

Over the first 15 months of maintenance and evolution work, the team has redocumented approximately 40% of the system. The accumulated documentation at that point covered approximately 60% to 70% of all new change requests. The change requests are concentrated in certain specific parts. Because of the opportunistic and incremental process of redocumentation, these parts were redocumented first.

About the Author



Václav Rajlich is a full professor and former chair in the Department of Computer Science at Wayne State University. His current research interests include software evolution and maintenance. He received a PhD in mathematics from Case Western Reserve University. He is a founder and past general chair of the IEEE International Workshop on Program Comprehension and past general chair of the IEEE International Conference on Software Maintenance. Contact him at the Dept. of Computer Science, Wayne State Univ., Detroit, MI 48202, rajlich@cs.wayne.edu, www.cs.wayne.edu/~vip/VadavRajlich.html

The company kept programmer and documentor timesheets for client billing. They included the activity code (such as study, coding, review, testing, or documentation), the start and end time, the module being maintained, the change request code, and so forth. These records showed that the cost of PAS is approximately 15% of the extra time. The cost is so low because the programmers must spend time in program comprehension anyway, and an extra 15% is spent to record the results in the PAS.

For this relatively small investment, the company gained the flexibility in scheduling people for the individual tasks. If the pending task dealt with annotated modules, the team no longer needed to wait for a specific person to complete his or her current task; rather, the task went to the first available programmer.

Training time for new programmers also fell, a significant reduction because over 15 months, seven new people joined the project, while five left. The PAS helped to transfer the program comprehension from the departing programmers to the new ones. The company found these benefits to be well worth the extra effort.

Future plans for PAS include addition of the automatic reporting system that will indicate which parts of the program have been redocumented, how often PAS is read and updated, how old are the PAS updates, and other indicators of the history of the PAS. These reports might help the managers to assess the redocumentation effort's effectiveness. ☛

References

1. R.K. Fjeldstad and W.T. Hamlen, "Application Program Maintenance Study: Report to Our Respondents," *Tutorial on Software Maintenance*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1982, pp. 13–30.
2. I.T. Bowman and R.C. Holt, "Reconstructing Ownership Architectures to Help Understand Software Systems," *Proc. Int'l Workshop Program Comprehension*, IEEE CS Press, 1999, pp. 28–37.
3. A. von Mayrhauser and A.M. Vans, "On the Role of Hypotheses During Opportunistic Understanding While Porting Large-Scale Software," *Proc. IEEE Workshop Program Comprehension*, IEEE CS Press, 1996, pp. 68–77.
4. M.A.D. Storey, F.D. Fracchia, and H.A. Muller, "Cognitive Design Elements to Support the Construction of a Mental Model during Software Visualization," *Proc. IEEE Int'l Workshop Program Comprehension*, IEEE CS Press, 1997, pp. 17–28.
5. V. Rajlich and S. Varadajan, "Using the Web for Software Annotations," *Int'l J. Software Engineering and Knowledge Engineering*, Vol. 9, No. 1, 1999, pp. 55–72.
6. A. Lakhota, "Understanding Someone Else's Code: An Analysis of Experience," *J. Systems and Software*, Vol. 23, 1993, pp. 269–275.