Mining Evolution Data of a Product Family*

Michael Fischer, Johann Oberleitner and Jacek Ratzinger
Distributed Systems Group
Information Systems Institute
Technical University of Vienna
A-1040 Vienna, Austria
{fischer,oberleitner,ratzinger}@infosys.tuwien.ac.at

Harald Gall
University of Zurich
Department of Informatics
s.e.a.l. – software
evolution & architecture lab
{gall}@ifi.unizh.ch

ABSTRACT

Diversification of software assets through evolving requirements impose a constant challenge on the developers and maintainers of large software systems. Recent research has addressed the mining for data in software repositories of single products ranging from fine- to coarse grained analyses. But so far, little attention has been payed for mining data about the evolution of product families. In this work, we study the evolution and commonalities of three variants of the BSD, a large open source operating system. The research questions we tackle are concerned with how to generate high level views of the system discovering and indicating evolutionary highlights. To process the large amount of data, we extended our previously developed approach for storing release history information to support the analysis of product families. In a case study we apply our approach on data from three different code repositories representing about 8.5GB of data and 10 years of active development.

1. INTRODUCTION

Unanticipated evolution of a single software system through changing requirements can lead to diversification and will result in different closely related products. These related products require a high maintenance effort which could be avoided by building a platform for a Product Family (PF) from existing assets. To identify candidates for building a software platform for a PF from related products, retrospective software evolution analysis can help to point out artifacts which exhibit a strong change dependency.

Most of the proposed mining approaches such as Zimmermann et al. for mining the change history [13] or Collberg et al. for visualizing a systems evolution [3] are justified to analyze data from a single source and would therefore need adaption to support data from multiple product variants. Analyzing a single product variant implies a strict order on historical information such as checkins into source code repositories. In contrast to this, multiple product variants can be roughly characterized through arbitrary and asynchronous release dates, unanticipated information flow between variants, different development goals and requirements.

Based on this conditions, with our *PfEvo* approach we address the problem of handling multiple, *asynchronously* maintained version control systems. The problem which has to be tackled first is the *comparability* of the different systems since they are frequently modified independent of each other. Thus release intervals are different, naming conventions, module structure, programming styles, etc. To obtain comparable results from different product variants, a fine grained time-scale—similar to sub-sampling—will facilitate synchronization of historical data from different source code repositories.

Artifacts with a strong change dependency frequently have architectural dependencies as research by Briand et al. has shown [1, 2]. Another frequently reason is duplicated code through *copy'n paste*. For the analysis of such change dependencies it would be beneficial if existing approaches and techniques could be adapted and reused to study their impact onto the module structure.

Based on results from change history analysis an expert may draw conclusions about commonalities of dependencies of the module structure. Then the identified software artifacts can be used as foundation for building a platform for a product family. A Representative of such a family of related products is the BSD operating system with its variants and derivations such as *NeXTStep*, *MacOS X*, or *SunOS*. In our case study we will use the free variants *FreeBSD*, *NetBSD*, and *OpenBSD* to show the applicability of our approach for data management and analysis.

In this paper we (1) apply and extend our approach [5] for extracting change history information and generating a release history database; (2) compare product variants on quantitative level for a coarse assessment of the historical development and assessment of the repository information for further research; and (3) apply our approach for the visualization of change dependencies [4].

The remainder of this paper is organized as follows: Section 2 presents our approach for studying product family evolution. In Section 3 we present our case study about three BSD variants. Section 4 presents related work and Section 5 draws conclusions and indicates future work.

2. AN APPROACH TO STUDY PRODUCT FAMILY EVOLUTION

Our *PfEvo* approach is an extension of existing techniques for the study of the evolution of a single software system and comprises the visualization of different aspects of the evolution of a software

*The work described in this paper was supported in part by the Austrian Ministry for Infrastructure, Innovation and Technology (BMVIT), the Austrian Industrial Research Promotion Fund (FFF), the European Commission in terms of the EUREKA 2023/ITEA project FAMILIES (http://www.infosys.tuwien.ac.at/Cafe/) and the European Software Foundation under grant number 417.

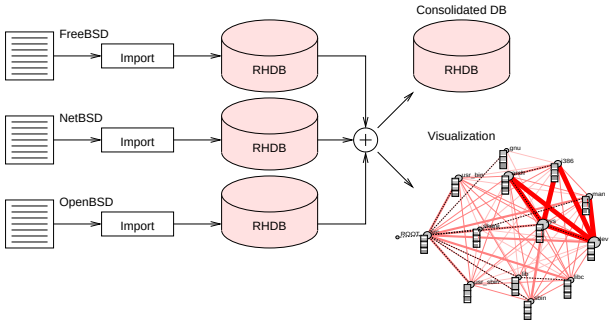

Figure 1: Process outline of *PfEvo*: results are a consolidated RHDB and visualizations

system. Besides some quantitative aspects such as the number of artifacts, checkin transactions, etc., these systems can be compared on a qualitative aspect as well. These quality aspects can be related to the type and extent of information flow between different systems, the impact of other related products on a single product, or hot-spots in the evolution of a single system with respect to information from other product variants.

To answer this research questions we have adopted our earlier approach for building a release history [5] and visualizing evolutionary information of large-scale software [4] and propose the process depicted in Figure 1. Since all data sources must undergo the same preprocessing steps—log file extraction, import into Release History Database (RHDB), detection of coupling groups, etc.—we use separate databases to store results. For subsequent analysis transactional data from the separate databases are filtered and merged into a new *consolidated* database which is better suited for queries spanning multiple product variants. Currently we use modified variants of existing queries to gather data from the three product databases. Another approach to compare system characteristics, is by visually comparing graphs describing a systems history. We will use: (1) histograms showing the distribution of change log data over the observation period; and (2) a graph indicating the impact of change dependency with respect to the module structure of the system.

In previous studies it was possible to use the release dates of the system under study as input for time scale information. Since the BSD variants are developed independently, an artificial, common time scale has to be created. This ensures comparableness of the different system histories. Disadvantageous is that is not possible to examine and compare the processes between the release dates, since the release intervals of the different product variants are cross-cut at arbitrary points. Since our requirement is the visualization of the resulting data-sets, we will use a sub-sampling interval of one month.

To detect and relate information flow between BSD variants we decided to use lexical search in change logs to find hints for information flow from other systems into the system under inspection. Alternatives to a pure lexical search are clone detection in source code, comparison of the structure of changes, or advanced indexing and text-analysis techniques.

3. CASE STUDY

For this case study we decided to use derivatives of the Berkley System Distribution also known as BSD Unix. The selected variants–*FreeBSD*, *NetBSD*, and *OpenBSD*–of BSD are large software systems consisting of an operating system kernel and a number of external programs such as *ls*, *passwd*, the GNU Compiler Collection (GCC), or the X windows system. The selected three variants have

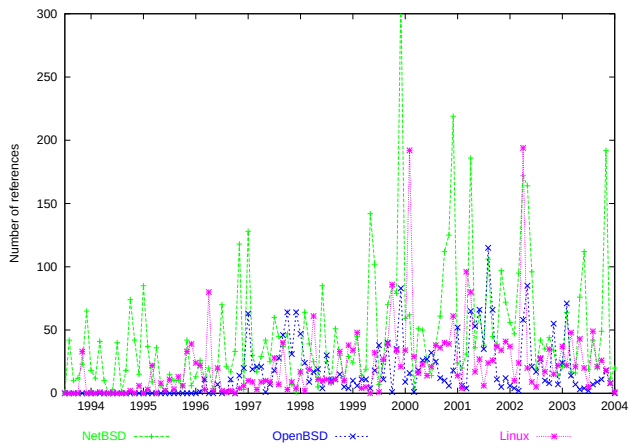

Figure 2: Number of references to keywords *NetBSD*, *OpenBSD*, and *Linux* found in *FreeBSD* change logs

between 4800 for the *OpenBSD* variant and 8000 directories for the *NetBSD* variant. The number of files varies between 30,000 (*FreeBSD*) and about 68,000 (*NetBSD*). They are long lived, actively maintained software systems representing about 8.5GB of data stored in three different repositories. Furthermore, release information is available as CVS [7] data for all three variants with direct access to the current repositories. The systems itself possess different characteristics which can be described as follows:

- The *FreeBSD*[1] projects aims to be more user application centric and thus it can seen as desktop OS rather than server platform. First release was in December 1993.
- *NetBSD*[2] is targeted onto portability and supports more than 10 different CPU types with together more than 50 different hardware platforms. Among them are exotic platforms such as *Acorn*, *Amiga*, *Atari* or *VAX*. First release was in October 1994.
- As representative of a server platform the aim of the *OpenBSD*[3] project lies on security and the integration of cryptography. First release was in October 1996.

While *NetBSD* and *FreeBSD* were directly derived from the *386 BSD 0.0* branch, *OpenBSD* was derived from the *NetBSD* branch in October 1995.

3.1 Quantitative comparison

First we give a quantitative comparison of the number of artifacts which are common for the different systems. To determine the number common C files in the different RHDBs we use a multi-database SQL queries. Table 1 shows the result for the different variants. While column "*all modules*" indicates the total number of common files found, column "*src/sys only*" indicates the common files within this subtree. Interesting is the high number of artifacts which are common in *NetBSD* and *OpenBSD*. This can be explained by the fact that *OpenBSD* was derived from *NetBSD* as mentioned previously.

[1]http://www.freebsd.org/ [31 December 2004]
[2]http://www.netbsd.org/ [31 December 2004]
[3]http://www.openbsd.org/ [31 December 2004]

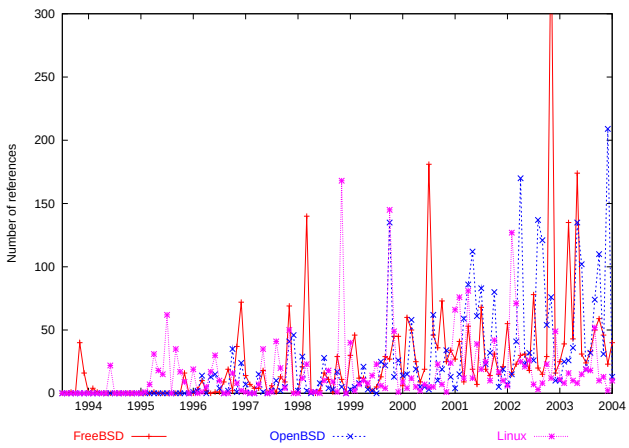

Figure 3: Number of references to keywords *FreeBSD*, *OpenBSD*, and *Linux* found in *NetBSD* change logs

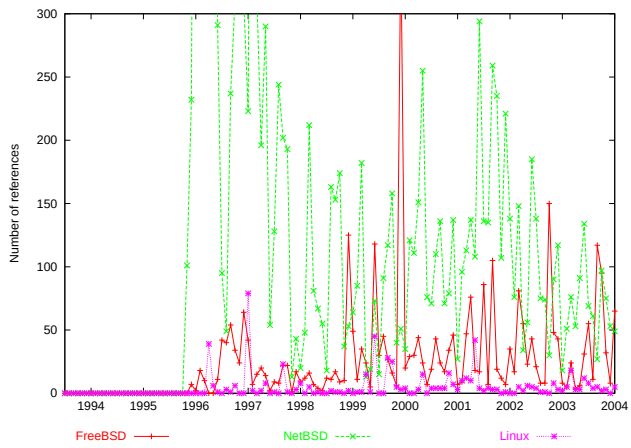

Figure 4: Number of references to keywords *FreeBSD*, *NetBSD*, and *Linux* found in *OpenBSD* change logs

Table 1: Common files in different BSD variants

Variant	Variant	all modules	src/sys/ only
FreeBSD	*NetBSD*	3810	1333
FreeBSD	*OpenBSD*	3839	1079
NetBSD	*OpenBSD*	6969	6847

3.2 Change report text analysis

As substitution for a detailed text and code clone analysis, we use keywords which were frequently used by the program authors and recorded in change reports. As useful keywords we identified *freebsd*, *netbsd*, *openbsd*, and interestingly *linux*.

Table 2: Information flow between variants of the BSD systems based on lexical search

Variant	Keyword	all revisions	revision > 1.1
FreeBSD	*netbsd*	5131	3577
FreeBSD	*openbsd*	2729	1353
FreeBSD	*linux*	1791	1387
NetBSD	*freebsd*	2852	2186
NetBSD	*openbsd*	2679	2224
NetBSD	*linux*	1547	1125
OpenBSD	*freebsd*	2406	1933
OpenBSD	*netbsd*	16802	7423
OpenBSD	*linux*	775	463

Table 2 lists the number of referenced artifacts between product variants based on a lexical search for the keywords *freebsd*, *netbsd*, *openbsd*, and *linux* in the change logs. Column one lists the name of the product variant used to retrieve the change logs and column two the respective keyword. Column three titled "*all revisions*" lists the number of distinct artifacts found in the RHDB having change logs with the specified keyword. Column four titled "*revision* > 1.1" lists the number of distinct artifacts found in the RHDB having change logs with the specified keyword and not having a revision number of "1.1" (which denotes the initial revision). The significant difference between the values in column three and four can be interpreted in such a way, that a larger number of files was imported from other systems and further maintenance is decoupled from the originating version.

3.3 Reference distribution

During the lexical search for the given keywords we recorded in total 12,540 change logs for *FreeBSD*, 9,468 for *NetBSD*, and 20,906 for *OpenBSD*. Based on this results, the Figures 2, 3, and 4 depict the distribution of references with respect to the observation period. Visually the histograms for *FreeBSD* and *NetBSD* indicate an increasing trend whereas the histogram for *OpenBSD* suggest a strong decreasing trend in the information flow from other platforms into *OpenBSD*. To underpin the visual perception of the trends we use linear regression analysis to find the dependency between the number of references and time-scale intervals.

Table 3: Linear regression for referenced keywords as $y = d + kx$ for the whole, for the years 1995–2001 ($y = d_{1,2} + k_{1,2}x$) and the years 2001–2004 ($y = d_{3,3} + k_{3,3}x$)

Variant	d	k	$d_{1,2}$	$k_{1,2}$	$d_{3,3}$	$k_{3,3}$
FreeBSD	22.7	0.897	-2.67	1.46	387	-2.35
NetBSD	-22.7	1.28	-15.7	1.14	-21.3	1.31
OpenBSD	407	-2.57	543	-4.90	668	-4.48

To test the development of the references over the given observation period we computed the values for the whole period and two sub-intervalls: the first interval accounts for about 2/3 (variables $k_{1,2}$ and $d_{1,2}$) of the observation period which corresponds to the years 1995–2001; the second interval accounts for about the last 1/3 (variables $k_{3,3}$ and $d_{3,3}$) of the observation period which represents the last 36 month of the development history or the years 2001–2004.

Table 3 shows the results for the three variants indicating a strong increasing trend for *FreeBSD* and *NetBSD* ($k > 0$ for both variants over the whole observation period). For *FreeBSD* this trend reverses for the last 36 month ($k_{3,3} < 0$). In contrast to this *OpenBSD* exhibits a decreasing trend in both sub-intervals and the whole observation period starting from a high level.

The low number of total change logs found for *NetBSD* and the positive trend in the change dependency of *NetBSD* suggest that large amounts of source code are still derived from other OS. This perception is also supported by Table 2 since *NetBSD* has the highest ratio between the two counted categories "*all revisions*" and "*revisions* > 1.1".

In the next sections we provide a more detailed look onto the change relationships with respect to different products.

3.4 Change impact analysis

To show the impact of changes onto the module structure with respect to foreign source code we selected *OpenBSD* for closer inspection since we counted here the most references from other OS

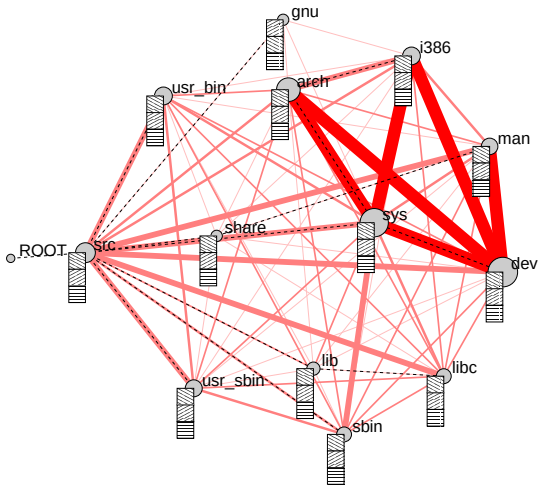

Figure 5: Change coupling between modules of the source code structure of the OpenBSD system

(see Table 2). The relevant artifacts were identified through lexical search as previously described. Based on the search results and the the change log data the impact of change dependencies on the module structure is evaluated. The result of this step is depicted in Figure 5. It shows the module structure together with change dependencies derived from the change log data. While filled circles indicate the nodes of the directory tree, shaded boxes indicate different product variants. We use ▧ as glyph for *FreeBSD*, ▨ for *NetBSD*, and ▤ is used for *Linux*. The approach for generating the layout for change dependences information is based on Multi Dimensional Scaling (MDS) [9] and has been used by our group to visualize to impact of problem report data onto the module structure of a large software system [4].

To avoid cluttering the figure with the several hundred modules of the source code package, we shifted relevant information from lower level nodes of the nested graph structure towards the root node until a predefined threshold criterion is met. The node sizes indicate the number of references found for each node and its subtrees.

While dashed lines indicate the directory structure of the source package, solid gray and black lines (pink and red on color displays) indicate the logical coupling between different parts of the system. Interesting to see is the strong logical coupling within the system related parts of *OpenBSD* such as *src/sys/arch/i386* and *src/sys/dev*. This results supports our expectations about the dependencies between different parts of the system. Easy to see are also the larger sizes of the nodes *src/sys/arch/i386*, *src/sys*, and *src/sys/dev*.

Table 4: Topmost referenced files with one of the given keywords in the change logs of *OpenBSD*

Keyword	Count	Path
freebsd	59	src/sys/dev/pci/files.pci
.	52	src/sys/dev/pci/pciide.c
.	52	src/sys/dev/pci/pcidevs
netbsd	45	src/sys/arch/i386/i386/machdep.c
.	43	src/sys/dev/pci/pciide.c
.	39	src/sys/conf/files
linux	14	src/sys/compat/linux/linux_socket.c
.	14	src/sys/compat/linux/syscalls.master
.	5	src/sys/dev/ic/if_wireg.h

Table 4 lists an excerpt of the topmost referenced artifacts which suggests a high information exchanges with other software systems. An example for the propagation of commonly required feature is the introduction of PCI devices. Since this bus type was not widely available at the time of the *OpenBSD* fork in 1996. Support for this bus type had to be added later requiring several separate changes as Table 4 suggests. Another interesting aspect is the relationship with *Linux*. The listing of *if_wireg.h* suggest that specific information about WLAN adapters are obtained from *Linux* as well.

3.5 Detailed change analysis

Since the three BSD variants originate from the same UNIX branch, it is to expect that also a number of source code changes exhibit the same or at least similar structure. For a manual verification we selected randomly one file which is available in all three variants. For this file–*ufs_quota.c* from the *src/sys/ufs/ufs/* directory–we manually inspected the revision history for significant changes.

One significant change was the modification of a function call in the *FreeBSD* version of *ufs_quota.c* on 1994-10-06 (revision 1.2 → 1.3) resulting in eight modified source lines. The *diff*-snippet–depicted below–for the affected source code revision shows a single change of a source line. The first line indicates the removed code, whereas the third one shows the replacement code. The three dashes in-between indicate a delimiter line.

```
< sleep((caddr_t)dq, PINOD+2);
---
> (void) tsleep ((caddr_t)dq, PINOD+2, "dqsync", 0);
```

In the change log we found the following comment, which indicates the reason for the source code modification: *"Use tsleep() rather than sleep so that 'ps' is more informative about the wait."*

The same modification in the *NetBSD* version has been applied on 2000-05-27 which is six years later than the original modification (revision 1.16 → 1.17) and in *OpenBSD* more than eight years later on 2001-11-21 (revision 1.7 → 1.8)–though without the *(caddr_t)* type cast listed in the preceding example. The *diff*-snippet below depicts the modification.

```
< sleep((caddr_t)dq, PINOD+2);
---
> (void) tsleep (dq, PINOD+2, "dqsync", 0);
```

In the *NetBSD* variant of the change log the comment is less informative: *"sleep() -> tsleep()"*. While in *NetBSD* this change still produces similar results when building the revision deltas via *diff*, in *OpenBSD* the change was part of a larger source code change consisting of 380 added and 161 deleted source lines (CVS does not identify modified lines, instead every modified lines accounts for one added and one deleted line). Similar to the given example, many changes can be found with varying degree of "identity" making it difficult to identify identical changes.

3.6 Discussion

During evaluation of our RHDB we noticed some shortcomings which have to be resolved prior to a thorough analysis of the different product variants. Firstly, through moving and renaming files in the CVS repository by the developers of the software systems, the historical information is segmented. Thus related segments have to be identified and concatenated to describe a continuous historical time-line of an artifacts history. Secondly, as result of the import process artifacts which have identical file names are assigned different IDs in the RHDB. This may adversarial effect multi-database queries for comparison of artifacts since artifacts with common origins have to identified for every evaluation of a database query.

This mapping of IDs will be ideally stored in the consolidated part of the RHDB as indicated in Figure 1.

From the software evolution analysis point of view, BSD represents an interesting software system which opens a wide field for further analysis. Since detailed information about the source code is available it would be beneficial to apply a tool for code clone detection such as [8] proposed by Kamiya et al. To improve the results of text analysis we currently evaluate the application of techniques related to Latent Semantic Indexing (LSI) [10].

4. RELATED WORK

Within the EU projects ARES, ESAPS, CAFE, and Families much work has been done in areas such as the identification of assets for product family architectures, evolution and testing of existing product families, architectural models for product families (Van der Linden [12]).

More related with our work is the approach with respect to product family evolution presented by Riva and Del Rosso in [11]. They investigated the evolution of a family platform and present approaches which enable architecture assessment and architecture reconstruction. In contrast to their work, we investigate the evolution of different variants to identify candidates for building a family platform.

In [6] Gall, Hajek and Jazayeri examined the structure of a *Telecommunications Switching Software* (TSS) over more than 20 releases to identify logical coupling between system and subsystems. This coupling is used in higher further processing steps to reveal evolutionary aspects such as hot-spots.

For the detection and visualization of evolutionary hot-spots we have developed a methodology which relates software feature and release history information [4]. In this paper we used information from the release history with respect to different keywords instead of feature data. This information was reflected onto the source base structure and visualized to generate the high level views of a software system.

5. CONCLUSIONS

Retrospective analysis of variants of related products opens interesting perspectives on the evolution of large software systems. With minimal changes and additions to existing tools it is already possible to recover the information flow between the different variants and evolutionary hot-spots with respect to the module structure. Through the application of a lexical search in the change logs we were able to reveal the increasing information flow of two variants of the systems. For the third system we found a decreasing flow starting from a very high level. For one selected system we applied an adapted method which generates high-level views of the module structure of system with respect to their coupling and information flow from other product variants. To support these findings about the information flow we performed detailed change analysis of a randomly selected file.

For future work we plan the application of a code clone detection process to identify related modifications. An analysis can reveal the degree and frequency how tight product variants are coupled. Another interesting area for future work is the detailed analysis of change log information for commonalities. Since change logs can provide additional hints about the origin of a particular information, they provide relevant information about the source of a particular change.

6. REFERENCES

[1] BRIAND, L., DEVANBU, P., AND MELO, W. An investigation into coupling measures for C++. In *Proceedings of the 19th international conference on Software engineering* (1997), ACM Press, pp. 412–421.

[2] BRIAND, L. C., DALY, J. W., AND WÜST, J. K. A Unified Framework for Coupling Measurement in Object-Oriented Systems. *IEEE Transactions on Software Engineering 25*, 1 (1999), 91–121.

[3] COLLBERG, C., KOBOUROV, S., NAGRA, J., PITTS, J., AND WAMPLER, K. A system for graph-based visualization of the evolution of software. In *Proceedings of the 2003 ACM symposium on Software visualization* (2003), ACM Press, pp. 77–ff.

[4] FISCHER, M., AND GALL, H. Visualizing Feature Evolution of Large-Scale Software based on Problem and Modification Report Data. *Journal of Software Maintenance and Evolution 16*, 6 (November/December 2004), 385–403.

[5] FISCHER, M., PINZGER, M., AND GALL, H. Populating a Release History Database from Version Control and Bug Tracking Systems. In *Proceedings International Conference on Software Maintenance (ICSM'03)* (September 2003), pp. 23–32.

[6] GALL, H., HAJEK, K., AND JAZAYERI, M. Detection of Logical Coupling Based on Product Release History. In *Proceedings International Conference on Software Maintenance* (March 1998), IEEE Computer Society Press, pp. 190–198.

[7] GRUNE, D., BERLINER, B., POLK, J., KLINGMON, J., AND CEDERQVIST, P. *Version Management with CVS*, 1992. http://www.cvshome.org/docs/manual/ [5 April 2004].

[8] KAMIYA, T., KUSUMOTO, S., AND INOUE, K. Ccfinder: A multilinguistic token-based code clone detection system for large scale source code. *IEEE Transactions on Software Engineering 28*, 7 (2002), 654–670.

[9] KRUSKAL, J. B., AND WISH, M. Multidimensional Scaling. *Quantitative Applications in the Social Sciences 11* (1978).

[10] LETSCHE, T. A., AND BERRY, M. W. Large-scale information retrieval with latent semantic indexing. *Information Sciences 100* (August 1997), 105–137.

[11] RIVA, C., AND DEL ROSSO, C. Experiences with software product family evolution. In *Proceedings Sixth International Workshop on Principles of Software Evolution (IWPSE'03)* (September 2003), IEEE Computer Society Press, pp. 161–169.

[12] VAN DER LINDEN, F., Ed. *Software Product-Family Engineering: 5th International Workshop, PFE 2003, Siena, Italy*, vol. 3014 of *Lecture Notes in Computer Science*. Springer-Verlag Heidelberg, 2004. http://www.springerlink.com/link.asp?id=HK53BTEN9EV9.

[13] ZIMMERMANN, T., WEISSGERBER, P., DIEHL, S., AND ZELLER, A. Mining Version Histories to Guide Software Changes. In *Proceedings 26th International Conference on Software Engineering (ICSE)* (May 2004), ACM Press, pp. 563–572.