Towards the Integration of CVS Repositories, Bug Reporting and Source Code Meta-Models

Giuliano Antoniol [1] Massimiliano Di Penta [2] Harald Gall [3]
Martin Pinzger [4]

[1,2] *RCOST - Research Centre on Software Technology*
University of Sannio, Department of Engineering
Palazzo ex Poste, Via Traiano 82100 Benevento, Italy

[3] *University of Zurich*
Department of Informatics

[4] *Technical University of Vienna*
Information Systems Institute

Abstract

Concurrent Versioning System (CVS) repositories and bug tracking systems are valuable sources of information to study the evolution of large open source software systems. However, being conceived for specific purposes, i.e., to support the development or trigger maintenance activities, they do neither allow an easy information browsing nor support the study of software evolution. For example, queries such as locating and browsing the faultiest methods are not provided.

This paper addresses such issues and proposes an approach and a framework to consistently merge information extracted from source code, CVS repositories and bug reports. Our information representation exploits the property concepts of the FAMIX information exchange meta-model, allowing to represent, browse, and query, at different level of abstractions, the concept of interest. This allows the user to navigate back and forth from CVS modification reports to bug reports and to source code. This paper presents the analysis framework and approaches to populate it, tools developed and under development for it, as well as lessons learned while analyzing several releases of Mozilla.

Key words: Source Code Analysis, Release History, Bug Reports, Object-Oriented Meta-Models

[1] Email: antoniol@ieee.org
[2] Email: dipenta@unisannio.it
[3] Email: http://www.ifi.unizh.ch/swe/
[4] Email: http://www.infosys.tuwien.ac.at/Cafe/

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

1 Introduction

The use of configuration management tools in software development and maintenance activities constitutes a consolidated practice, also supported by software engineering principles. Versioning systems such as the Concurrent Versioning System (CVS) are commonly adopted, especially for large-scale projects. Often versioning system are complemented by bug reporting tools, that constitute an essential support for corrective maintenance.

These two families of tools are valuable sources of information to study software evolution; CVS can be exploited to get insights about evolution in terms of size, complexity, amount of changes, and whatever can be mined from source code or from change logs. On the other hand, bug reporting systems provide insights on reliability, as well as information on how an organization manages defects (e.g., what is the average defect fixing rate, statistics about defect severity).

CVS repositories and bug tracking systems are almost always not integrated; integration would allow maintainers and project managers to get an overall view. For instance, it would be feasible to identify a subset of components (classes, functions or methods above a given size) which exhibit more defects than others. Similarly, one could analyze the relationship between defects and some language constructs e.g., the use of pointers or inheritance versus defect proneness.

It is the opinion of the authors that the task of integrating heterogeneous sources of information may be simplified by the adoption of a meta-model allowing to accommodate source code abstractions, source code changes as well as details on bug reports. Different schemata and meta-models have been proposed to represent both procedural and object-oriented (OO) software; however, the proposed schema either have been tailored to a specific language such as C, C++, Smalltalk or Java, or have not been annotated with source code level details nor integrated with other relevant source of information such as bug reports and CVS data.

This paper proposes to enrich the FAMIX information exchange meta-model [19] with detailed information extracted from a combination of heterogeneous sources, namely source code, CVS and bug tracking repositories. FAMIX provides the concept of property which naturally leads to decorate entities of interest (e.g., classes or methods) with a variety of details. Clearly, entity decoration depends on the particular entity. Some decorations, such as file name and subsystem name, are common to all entities. Changes and defects are decorations applicable class, method, and function entities. Finally, other properties, such as the number of parameters, belong exclusively to templates, methods and functions.

To reason about changes, bugs and source code entities, to relate facts or extract statistics, we must build a traceability map between different concepts. At this purpose, we follow an approach inspired by the `diff` and `patch` Unix utilities. Changes are identified by means of file name and line numbers, hereby referred as *location by site*. This permits the integration of the information extracted from source code, bug reports and CVS change logs. The above choice stems from the following observation. Modification Reports (MR) often detail involved files

and changed lines of code; once files and changed lines have been identified, the changed context (class and method) can be located by parsing sources extracted from a CVS repository. Vice-versa, given a contextual information (file, beginning and ending line) it is possible to identify the a Problem Report (PR) impacting that code region.

To verify feasibility, discover particular strengths, problems and pitfalls, the framework was applied to several releases of Mozilla. Mozilla is a multi-million LOCs open source project [5] managed via a CVS repository and a bug tracking system. Source code information, CVS information and bug information were used to decorate our instantiated meta-model. Particularly, releases from 1.0 to 1.3.1 were used as case study. At the time of writing, integration has been achieved at the file level; more fine grained integration, namely at the class, method or line level is only partially supported.

The main contribution of this paper can be summarized as follows:

- we provide an extension of RSF to decorate the FAMIX meta-model;
- we propose an integrated framework allowing the user to access bug reports, CVS, and source code facts;
- we discuss lessons learned in populating a repository with several releases of Mozilla.

The remainder of the paper is organized as follows. After a review of the related work, Section 3 proposes our approach and presents the framework. Section 4 describes the Mozilla case study. Section 5 describes our tools developed to extract information and to populate the repository; Section 6 reports on and summarizes the experiences gained in populating the first release of the repository. Finally Section 7 concludes and outlines foreseeable research activities.

2 Related Work

There is not so much work in literature related to integrate and analyze bug reporting and release history information. The present work stems from the release history database of Fisher et al. [8]. The authors proposed to combine CVS revision data with bug reporting data and to add some missing information such as, for example, merge points. The same authors also performed, on the same data, an analysis devoted to track features [7]. Finally, Gall et al. [12] analyzed CVS release history data for detecting logical coupling.

In [16] maintenance requests were classified according to maintenance categories as IEEE Std. 1219 [1], then rated on a fault-severity scale using a frequency-based method. Ball et al. [5] proposed an approach for visualization of data extracted from a version control system, while a three-dimensional color visualization of release history was proposed in [13].

Some other relevant studies have been performed in the past with the purpose of

[5] `http://www.mozilla.org`

understanding the architecture and the evolution of the Mozilla open source project. In particular, Godfrey et al. integrated different reverse engineering tools and used them to extract Mozilla's architecture [14]. Mockus et al. studied the evolution of two large open source projects, namely Apache and Mozilla [15].

3 The Framework

The main idea adopted to locate, browse, and integrate heterogeneous information is to rely on *location by site*: classes, methods, functions as well as defects and changes must be located by file and line number. CVS systems keep track of changes; parsing source code locates classes, methods, functions and other source code level concepts. Bug reports are semi-structured documents; they often contain details on impacted files and lines.

The following subsections provide details on the approaches, implemented or under development, adopted to build traceability mapping between code area regions (classes, methods, functions), PRs, and MRs.

At the time of writing sources code, PRs, and MRs are managed at different granularity levels. Integration at file level has already been achieved; browsing at class, method or line of code level is only supported from source code entity to code region. Another limitation of the current implementation is the impossibility of performing time related queries (e.g., locating classes modified before a given date).

3.1 Bugzilla and CVS repositories

Release history data is retrieved from versioning systems such as the CVS [10] and bug tracking systems such as Bugzilla [22]. In particular, we obtain MRs from CVS and PRs from Bugzilla. Figure 1 depicts the nucleus of our *Release History Database* with linked MRs and PRs. MRs are stored in the *cvsitemlog* entity and PRs in the *bugreport* entity of our RHDB. Information about the file to which a MR belongs is stored in *cvsitem*.

Links between MRs and PRs are stored to the table *cvsitemlogbugreport*. Establishing such links is an important issue of the RHDB population process. Concerning CVS and Bugzilla, this needs to be done separately. A link is stored whenever a reference to a PR is found in a MR. PR figures in MRs are searched using regular expressions (e.g., #128764). Because these numbers are entered as free text, results contain correct and false positive matches as well. To improve data quality, all matched numbers are validated using information available with PRs such as patches that contain the names of files they are applied to. If this file name corresponds to the name of the file of the MR the link is validated [7]. More details on bug report and CVS processing can be found in [7,8,12]

Summarizing, the RHDB contains versioning, change and defect relevant data about each file of each release that has to be integrated with source model data as described next.

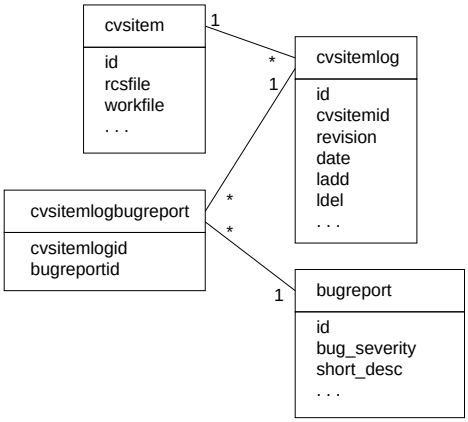

Fig. 1. “Nucleus” of the RHDB

3.2 Source Code Modeling

All the concepts available at design level, such as those modeled via UML diagrams, function invocation, and software metrics are extracted and represented. The source code meta-model was inspired by the FAMIX [19] information exchange meta-model [6] .

FAMIX prescribes CASE Data Interchange Format (CDIF) [6] as the basis for information exchange. Other standards and interchange formats exist, for example XMI, an XML based interchange format [18], or the Rigi Standard Format (RSF) [20]. The RSF origins from the Rigi program visualization, reverse engineering and program understanding environment, and is a triple based specification language that can be easily customized and imported into different tools. Figure 2 shows an RSF excerpt of a class representation. We use RSF to represent the concepts of interest such as classes, class attributes and methods, types, or different kinds of software metrics, etc.

Figure 3 shows, at a high level of abstraction, the steps carried out and the extracted information for the Mozilla browser. To avoid problem of missing files, wrong dependencies and compilation errors, the approach is a two phase approach. First, the source code undergoes a preliminary compilation to produce the target executables. Then source code is parsed and information extracted.

The two phases approach ensures that the application is properly configured for the current instance of architecture, operating system and, in general, hardware and software environment. Clearly, when PRs refer to configuration-dependent source code, consistency needs to be ensured between PRs and the source code facts. This paper focuses on source code (and thus PRs) related to a single configuration, for a Linux operating system and Intel architecture.

To reuse already available tools, the extracted information is first represented via an extension of an intermediate language, name Abstract Object Language (AOL). AOL is a general-purpose design description language capable of express-

[6] `http://www.iam.unibe.ch/~famoos/`

```
type nsAutoRefCnt "Class"
contain ./dist/include/xpcom/nsISupportsImpl.h
                  nsAutoRefCnt
lineno  nsAutoRefCnt    "88"

type nsAutoRefCnt::mValue "Attribute"
belongsToClass  nsAutoRefCnt::mValue
         "nsAutoRefCnt"
type nsrefcnt "DataType"
hasType nsAutoRefCnt::mValue "nsrefcnt"
accessControlQualifier
         nsAutoRefCnt::mValue "PRIVATE"

type nsAutoRefCnt::nsAutoRefCnt() "Method"
isAbstract nsAutoRefCnt::nsAutoRefCnt() "TRUE"
belongsToClass  nsAutoRefCnt::nsAutoRefCnt()
         "nsAutoRefCnt"
type nsAutoRefCnt "DataType"
hasType nsAutoRefCnt::nsAutoRefCnt()
         "nsAutoRefCnt"
accessControlQualifier
         nsAutoRefCnt::nsAutoRefCnt() "PUBLIC"
```

Fig. 2. Excerpt of a class RSF representation

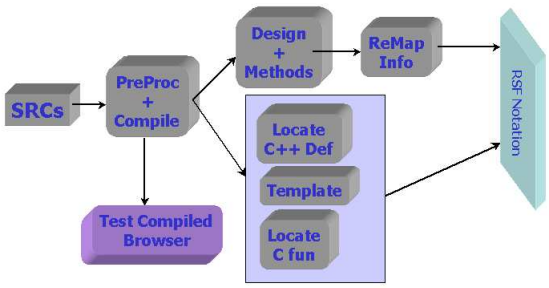

Fig. 3. C++ information extraction

ing concepts available at the design stage of OO software. It has been extended to represent software metrics, structures, templates and other facts such as methods or functions calls. More details on AOL can be found in [2,3,4,9]. Once produced, the AOL is scrutinized, if needed, verified, and finally mapped into the customized RSF final representation.

Figure 4 details the steps encompassed by the box *PrepProc + Compile* of Figure 3. The second compilation relies on wrappers wrapping C and C++ compilers. This is needed to avoid error prone activities required to modify by hand compilation scripts and makefiles. The preprocessed files contain both application and system information. Thus a further step may be needed to get rid of unusable information, i.e., to remove system include files.

Summarizing, RSF information comprises:

- a reverse engineered class diagram including inheritance, association and aggregation relationships;

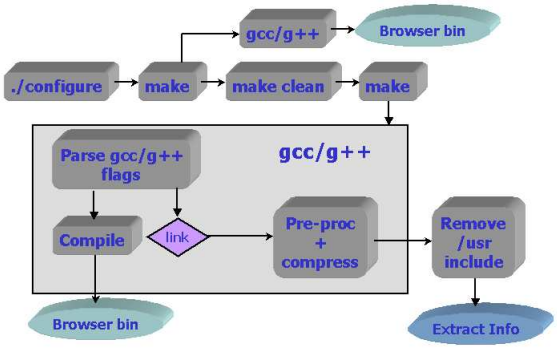

Fig. 4. C++ processing

- function and method level software metrics such as the number of passed parameters, the maximum nesting level, or the number of statements;
- details on template and structures; and
- location by site of classes, methods and functions.

3.3 Integrating the Information Sources

For the integration of source model and release history data, an entity common to both information spaces has to be determined. As already stated, in the context of this paper, we focus on source files to be the common entity because files are subject of versioning, change and defect data as retrieved from versioning and bug reporting systems. A finer-grained level of integration is under development and subject of ongoing work.

In our approach, source model information is available in FAMIX conform RSF files. Release history information is stored in a relational database and can be queried using SQL. The key connector that links both information sources is the unique name of files contained in both repositories. Based on these unique file names the data integration process is performed. The process is straightforward, and it consists of the two steps: 1) query RHDB and output results in RSF; 2) integrate results with source model RSF files.

In the first step we query the RHDB database with respect to the dependencies due to MR and PR data. The input to the query is a list of unique names of source files of interest. According to Figure 1, we query the `cvsitem` table to get the file identifiers which on their own, are used to query the `cvsitemlog` joined with `cvsitemlogbugreport` and `bugreport` relations to get change related dependencies between selected source files.

Regarding both MR and PR, we introduce two new relationship types `rhdbCoupled` and `rhdbDependent`. The fundamental principle of both relationship types is that from the point of view of changes two source files are *logically coupled* or *dependent* if they have been affected by the same source code modification [11].

Consequently, if two source files have been checked into the source repository (i.e., CVS) at about the same time they are logically coupled. Furthermore, if two files are referenced by the same PR, then from the point of view of changes they are dependent. Additionally, we compute the number of affected MRs and PRs to determine the weight of these relationships. Results of the queries are output in the form of RSF tuples.

The second step of the integration process is concerned with integrating computed RSF tuples to the source model data. This includes determining affected source files in the source model database and computing the new edge identifiers for each integrated `rhdbCoupled` and `rhdbDependent` relationship.

The result of the integration process comprises a source model repository enriched with release history data that, on its own, comprises `rhdbCoupled` and `rhdbDependent` relationships between source file entities, as well as a `weight` attribute for each relationship.

4 The Case Study

To evaluate the feasibility of integrating source code level and quality related information, several versions of Mozilla, an open source web browser, were analyzed. The extracted key features are reported in Table 1. Mozilla was mostly developed in C++; the C code accounts only for a small fraction of the overall size. XML, HTML and scripting language configuration and support programs are also present. The latest Mozilla releases include more than 10,000 source files for a size up to 3.7 MLOC located in 2,500 subdirectories. Mozilla basically consists of 90 modules maintained by 50 different module owners. The Bugzilla bug tracking system contains more than 180,000 PRs and the CVS repository contains about 430,000 MRs.

Release	# C files	# h files	# C++ files	Size Size	Classes Classes	Methods Methods	Func. Func.	Inheritances	Associations	Aggregations
1.0	1987	7,519	3,982	3.5	4,545	50,912	5,737	5,031	6,993	3,404
1.0.1	1995	7,603	4,022	3.5	4,561	54,742	5,740	5,051	7,006	3,440
1.0.2	1987	7,635	4,049	3.5	4,572	51,198	5,740	5,065	7,029	3,461
1.1	1997	7,674	4,054	3.6	4,594	52,453	5,742	5,095	7,048	3,466
1.2a	1984	7,769	4,058	3.6	4,475	51,104	5,741	4,992	7,107	3,514
1.2b	1991	7,972	4,122	3.7	4,512	53,697	5,794	5,029	7,141	4,804
1.2	1991	7,981	4,129	3.7	4,526	51,689	6,192	5,044	7,155	4,817
1.2.1	1991	7,981	4,129	3.7	4,524	52,953	5,794	5,044	7,156	4,817
1.3a	1823	7,880	4,145	3.6	4,574	51,827	5,809	5,081	7,157	6,090
1.3b	1830	7,924	4,164	3.6	4,589	51,580	5,836	5,101	7,339	6,200
1.3	1830	7,911	4,158	3.6	4,577	53,106	5,836	5,088	7,323	6,181
1.3.1	1830	7,935	4,198	3.7	4,577	51,453	5,836	5,088	7,323	6,181

Table 1
Mozilla releases key features

5 The Tools

Several tools were reused, modified or developed to extract and integrate information from the different sources.

5.1 Compiler Wrappers

C and C++ compiler wrappers, mimicking the same compiler interface, have been developed with Perl. `gcc` and `g++` specific options as well as linker options are fully supported and managed. Preprocessed source code is compressed to reduce disk space usage.

5.2 C++ Information Extraction

A tool inspired by island-driven parsing [17] has been reused and modified to reverse engineer a class diagram and extract class level and method level metrics. The island-parsing approach allowed to overcome most of the difficulties related to parsing C++ code (intrinsic language difficulties, dialects such as the GNU dialect encountered when parsing *Mozilla*, etc.). The tool was developed in previous projects to extract AOL. More details can be found in [2,3,4,9].

5.3 FAMIX Export and RSF Integration

The exporter and integration tool comprises two Perl scripts that process AOL files and output FAMIX data in RSF, as well as integrate the data of two RSF files into one. AOL is the format used by the C/C++ parser to output extracted facts. Although all the extracted information is present, AOL lacks of representing the information in a graph-like manner that is used by source model analysis and visualization tools. Thus, we used the extension points of the FAMIX model to specify additional attributes that hold the various metrics extracted for each source model entity. Furthermore, we also added two new relationships between source files to store logical couplings `rhdbCoupled` and hidden dependencies `rhdbDependend`.

Two steps are performed by the exporter, namely: 1) mapping AOL to RSF data; 2) integrating RSF files into one source model data file. Basically, the parser generates separate AOL files for storing extracted information about source files, classes, methods and functions. In the first step each of these files are input to the exporter that prints out plain RSF tuples of contained information. Preliminary checks are performed that consider the data stored in a files. For example class and inheritance relationships of the AOL class file are checked to existence of the base and subclass. AOL records that fail the check are not printed. In the second step the different RSF files are integrated into on file that contains the whole source model. During this integration process, existence checks on entities of relationships and attribute records are performed but in this step checks involve the whole data source.

Data about logical couplings and hidden dependencies between source files are also available in RSF format. Hence, the integration script is applied to add this relationships and the weight attribute to the existing source model data. The output is a RSF file that contains the integrated source model data in FAMIX conform RSF tuples which can be handled by existing visualization tools such as Rigi [21] or SHriMP [23].

6 Lesson Learned

This section summarizes lessons learned while integrating bug reports, CVS information and data from 12 Mozilla releases.

Mozilla is a large software system, encompassing a variety of programming languages, styles and idioms. If a language contains a feature, someone will use it regardless of the impact it could have on understandability, portability maintainability, or evolvability. For example, C++ is a strongly typed, OO language. However, it retains C compatibility and considers a `struct` as a `class` only containing public attributes. This means that there may be classes inheriting from structs e.g., `struct nsBandData` and `class nsBlockBandData`. It should be noted that this is something different from wrapping a structure with a class, since it breaks information hiding and encapsulation. `nsBandData` is declared as a structure and there seems to be no reason why it should not be declared as a class but allowing access from C. In fact, a C++ compiler, compiles C code if it does not break C++ rule e.g., a `new` identifier causes compilation failure. However, C and C++ compilers have different conventions and thus the above practice may easy the task to break C++ encapsulation from C code.

Much in the same way, we observed structures containing method declarations. For example, `nsID` is a structure with three functions declared inside one of which is both declared and defined inside `Equals`. Again, there is no obvious reason but allowing an easier access from C code.

Templates constitutes a powerful mechanism to enable for parametric code development. However, authors believe that in certain cases it may also be abused. While, templates should be used to create new *abstract data types*, weird uses of templates were found in Mozilla. For example, we found examples of templates, e.g., `nsCOMTypeInfo`, used to parameterize a structure.

These latter peculiar uses or *abuses* opened a discussion on what should be annotated on the meta-model, if and when the meta-model should be emended. The information is available; however it is not completely clear if and how certain facts have to be represented. For example, the `struct nsBandData` could be represented as a class and then flagged as an OO coding style violation or we have to extend FAMIX so that a class may inherits from structs. On the other hand, it should be noted that coding style violation are not usually part of source code meta-models. All in all, it further led to the need to modify the initial integration model for example to cope with classes derived from structures or structures with methods.

7 Conclusions and work-in-progress

Consistently integrating different repositories of large software systems, such as the open source software Mozilla and its CVS and Bugzilla data, is a challenging but fruitful task. It allows a user to represent, browse and query—at different levels of abstraction—the particular concept of interest, from the source code level to the bug report and modifications level. In this paper, we proposed a first step toward a *multi-level concept navigation framework* that represents source entities in FAMIX meta-model compliant Rigi Standard Format (RSF).

We took the Mozilla browser as a case study and extracted its C++ sources over 12 releases into an RSF representation. This kind of data was combined with release history data populated from filtering related bug reports and modification reports into a release history database (RHDB). This release data integration venture allowed us to highlight problems encountered, difficulties and pitfalls.

Work–in–progress is devoted to manage the discovered information overflow problem, and complete the integration, ensuring a finer level of detail (i.e., tracing problems to classes and methods) and to ensure appropriate querying and browsing capabilities.

8 Acknowledgments

This research was partially supported by the RELEASE Excellence network founded by the European science Foundation.

References

[1] "IEEE std 1219: Standard for Software maintenance," 1998.

[2] Antoniol, G., B. Caprile, A. Potrich and P. Tonella, *Design-code traceability for object oriented systems*, The Annals of Software Engineering **9** (2000), pp. 35–58.

[3] Antoniol, G., G. Casazza, M. Di Penta and R. Fiutem, *Object-oriented design patterns recovery*, Journal of Systems and Software **59** (2001), pp. 181–196.

[4] Antoniol, G., R. Fiutem and L. Cristoforetti, *Using metrics to identify design patterns in object-oriented software*, in: *Proceedings of 5th International Symposium on Software Metrics - METRICS98*, Bethesda MD, 1998, pp. 23–34.

[5] Ball, T., J. M. Kim, A. Porter and H. Siy, *If your version control could talk ...*, in: *ICSE Workshop on Process Modeling and Empirical Studies of Software Engineering*, Boston, MA, USA, 1997.

[6] Committee, C. T., "CDIF Framework for Modelling and Extensibility," Electronic Industries Association EIA/IS-107, 1994.

[7] Fischer, M., M. Pinzger and H. Gall, *Analyzing and Relating Bug Report Data for Feature Tracking*, in: *10th Working Conference on Reverse Engineering (WCRE), Victoria, Canada*, 2003, pp. 90–99.

[8] Fischer, M., M. Pinzger and H. Gall, *Populating a release history database from version control and bug tracking systems*, in: *Proceedings of IEEE International Conference on Software Maintenance*, Amsterdam, The Netherlands, 2003, pp. 23–32.

[9] Fiutem, R. and G. Antoniol, *Identifying design-code inconsistencies in object-oriented software: A case study*, in: *Proceedings of IEEE International Conference on Software Maintenance*, Bethesda MD, 1998, pp. 94–102.

[10] Free Software Foundation, "Version Management with CVS," 1.11.14 edition (2003), http://www.cvshome.org/docs/manual.

[11] Gall, H., K. Hajek and M. Jazayeri, *Detection of logical coupling based on product release history*, in: *Proceedings of the International Conference on Software Maintenance (ICSM '98)* (1998).

[12] Gall, H., M. Jazayeri and J. Krajewski, *Cvs release history data for detecting logical couplings*, in: *Proceedings of the Interational Workshop on Principles of Software Evolution*, Helsinki, Finand, 2003, pp. 13–23.

[13] Gall, H., M. Jazayeri and C. Riva, *Visualizing software release histories: The use of color and third dimension*, in: *Proceedings of IEEE International Conference on Software Maintenance*, Oxford, England, 1999, pp. 99–108.

[14] Godfrey, M. and E. Lee, *Secrets from the Monster: Extracting Mozilla's Software Architecture*, in: *Proceeding of Second Symposium on Constructing Software Engineering Tools*, 2000.

[15] Mockus, A., R. Fielding and J. Herbsleb, *Two case studies of open source software development: Apache and Mozilla*, ACM Transactions on Software Engineering and Methodology **11** (2002), pp. 309–346.

[16] Mockus, A. and L. Votta, *Identifying reasons for software changes using historic database*, in: *Proceedings of IEEE International Conference on Software Maintenance*, San Jose, California, 2000, pp. 120–130.

[17] Moonen, L., *Generating robust parsers using island grammars*, in: *Working Conference on Reverse Engineering*, 2001.

[18] OMG, "XML Metadata Interchange (XMI)," OMG Document ad/98-10-05, 1998.

[19] Software Composition Group, University of Berne, "The FAMIX 2.0 specification," 2.0 edition (1999), http://www.iam.unibe.ch/ scg/Archive/famoos/FAMIX/.

[20] Tilley, S. R., K. Wong, M.-A. D. Storey and H. A. Müller, *Programmable reverse engineering*, International Journal of Software Engineering and Knowledge Engineering **4** (1994), pp. 501–520.

[21] Wong, K., S. Tilley, H. A. Muller and M. D. Storey, *Structural redocumentation: A case study*, IEEE Software (1995), pp. 46–54.

[22] *Bugzilla Bug Tracking System*, http://www.bugzilla.org.

[23] *Shrimp views: Simple hierarchical multi-perspective*, http://shrimp.cs.uvic.ca/ (2004).