KUOPION YLIOPISTON JULKAISUJA H. INFORMAATIOTEKNOLOGIA JA KAUPPATIETEET 9
KUOPIO UNIVERSITY PUBLICATIONS H. BUSINESS AND INFORMATION TECHNOLOGY 9

TIMO KOPONEN

Evaluation of Maintenance Processes in Open Source Software Projects Through Defect and Version Management Systems

Doctoral dissertation

To be presented by permission of the Faculty of Business and Information Technology of the University of Kuopio for public examination in Auditorium, Mediteknia building, University of Kuopio, on Friday 21st September 2007, at 12 noon

Department of Computer Science
University of Kuopio

KUOPION YLIOPISTO
KUOPIO 2007

Distributor: Kuopio University Library
P.O. Box 1627
FI-70211 KUOPIO
FINLAND
Tel. +358 17 163 430
Fax +358 17 163 410
www.uku.fi/kirjasto/julkaisutoiminta/julkmyyn.html

Series Editors: Professor Markku Nihtilä, D.Sc.
Department of Mathematics and Statistics

Assistant Professor Mika Pasanen, D.Sc.
Department of Business and Management

Author's address: Department of Computer Science
University of Kuopio
P.O. Box. 1627
FI-70211 KUOPIO
FINLAND

Supervisors: Development and research manager Virpi Hotti, Ph.D.
Department of Computer Science
University of Kuopio

Reviewers: Professor Markku Tukiainen, Ph.D.
Department of Computer Science and Statistics
University of Joensuu

Docent Jussi Koskinen, Ph.D.
Department of Computer Science and Information Systems
University of Jyväskylä

Opponent: Professor Reijo Sulonen, Ph.D.
Software Business and Engineering Institute
Helsinki University of Technology

ISBN 978-951-781-988-6
ISBN 978-951-27-0107-0 (PDF)
ISSN 1459-7586

Kopijyvä
Kuopio 2007
Finland

Koponen, Timo. Evaluation of Maintenance Processes in Open Source Software Projects Through Defect and Version Management Systems. Kuopio University Publications H. Business and Information Technology 9. 2007. 92 p.
ISBN 978-951-781-988-6
ISBN 978-951-27-0107-0 (PDF)
ISSN 1459-7586

ABSTRACT

While Open Source Software (OSS) is becoming more widespread and popular, its maintenance has become an important issue. However, several aspects of Open Source Software maintenance are unclear; for example, maintenance processes are usually described briefly in the project documentation and the descriptions are very informal and inaccurate. Several earlier studies have examined Open Source Software projects (OSSPs) but they have not provided information about maintenance because they have concentrated on several other aspects such as development and community models. Therefore, in this thesis we study the maintenance of Open Source Software.

First, we present an *evaluation framework for Open Source Software* that can be used to create a general overview of the software, project and related services. From the viewpoint of the software, we cover attributes such as the type of software and intended audience. From the project viewpoint, we cover aspects of the development model and management systems. Since most OSSPs do not provide services to users, it is reasonable to cover external service providers.

Second, we present an *Open Source Software maintenance process framework*. It was developed by studying maintenance activities and the project documents in OSSPs and using ISO/IEC and IEEE standards for software maintenance as a guideline. As a result of this, we noticed that from the theoretical viewpoint the maintenance processes in OSSPs are quite similar to those in the standards. However, maintenance activities may not be performed as described in the project documents. Therefore, we decided to identify attributes and metrics that could be used to measure maintenance processes in the real world.

Third, we present an *evaluation framework for Open Source Software maintenance* that can be used to study OSS maintenance processes through defect reports and software change histories by using defect management systems (DMS) and version management systems (VMS) as data sources. While those systems provide more than enough data about maintenance processes, data retrieval can become a problem. One OSSP can have over 35000 defect reports and 7000 changes in the source code, so manual retrieval and evaluation of the data is not a realistic option. Therefore, we developed an automatic tool, Remote analysis System for Open Source Software (RaSOSS), for data retrieval and analysis.

The main contributions of this thesis are these three frameworks and RaSOSS.

Universal Decimal Classification: 004.413, 004.415.5

Inspec Thesaurus: computer software; public domain software; software maintenance; software management; project management; system documentation; software standards; error analysis; error handling; data analysis

Acknowledgements

It gives me much pleasure to conclude a few years of work by expressing my gratitude to individuals and institutions that have supported the research presented in this thesis. I would like to thank my supervisor Virpi Hotti for her guidance and motivation. I wish also to thank the Department of Computer Science at Kuopio University and all colleagues.

Finally, I also want thank my parents Eva and Esa, and my dear wife Sanna.

(Almost) Anything is possible; just will and hard work is needed.

Kuopio 21.9.2007

_____

Timo Koponen

ORIGINAL PAPERS

Paper I: Koponen T, Hotti V. Evaluation Framework of Open Source Software. *Proceedings of The International Conference on Software Engineering Research and Practice SERP'04, Vol. II.*, Las Vegas, Nevada, USA, June 21-24, 2004, pp. 897-902. CSREA Press, 2004.

Paper II: Koponen T, Hotti V. Open Source Software Maintenance Process Framework. *Proceedings of The Fifth Workshop on Open Source Software Engineering,* St. Louis, USA, 17.5.2005, ACM SIGSOFT software engineering notes Vol. 30. Issue 4, pp. 1-5. New York: ACM Press, 2005.

Paper III: Koponen T, Hotti V. Defects in Open Source Software Maintenance - Two Case Studies: Apache and Mozilla. *Proceedings of The 2005 International Conference on Software Engineering Research and Practice SERP'05, Vol. II*., Las Vegas, Nevada USA, June 27-30, 2005, pp. 688-693. CSREA Press, 2005.

Paper IV: Koponen T. Life Cycle of Defects in Open Source Software Projects. *IFIP International Federation for Information Processing 203, Open Source Systems; Proceedings of the 2nd International Conference on Open Source Systems (OSS 2006) (IFIP Working Group 2.13 Foundation on Open Source Software)*, Como, Italy. June 8-10, 2006. pp. 195-208, Springer.

Paper V: Koponen T., Lintula H.: Are the Changes Induced by Defect Reports in Open Source Software Maintenance? *The 2006 International Conference on Software Engineering Research and Practice, Vol I.,* June 27-30, 2006, Las Vegas, USA. pp. 429-435, CSREA Press.

Paper VI: Koponen T.: Evaluation Framework for Open Source Software Maintenance. *The International Conference on Software Engineering Advances (ICSEA'06).* October 2006, Tahiti, French Polynesia. pp. 52. IEEE Press.

Paper VII: Koponen T.: RaSOSS - Remote Analysis System for Open Source Software. *The International Conference on Software Engineering Advances (ICSEA'06).* October 2006, Tahiti, French Polynesia. pp. 54. IEEE Press.

Table of Contents

1 INTRODUCTION

Open Source Software (OSS) is becoming increasingly important and widespread. OSS covers a wide range of software types from desktop software such as Open Office and Mozilla Firefox, to Enterprise Resource Planning (ERP) such as Compiere and ERP5. These software are used in countless companies, governments, communities and homes.

The major difference between OSS and proprietary software is that OSS has been developed within a community and licensed with a license that has been approved by the Open Source Initiative (OSI) [OSI06]. OSI approved licenses give users the rights to use, redistribute and modify software. These rights make distributed software development possible, where almost anyone can participate and help to develop the software.

However, distributed development and free participation is fundamentally different from software engineering, which is defined in the IEEE Standard Glossary of Software Engineering Technology (IEEE 610.12) as [IEE04]:

> "The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software, that is, the application of engineering to software."

Whereas software engineering uses a systematic and strictly controlled approach, OSS projects (OSSP) use a more liberal approach, where almost anyone can participate in project activities such as defect reporting, development and maintenance. Because of the free participation and large number of developers, OSSPs use several systems for management. These management systems, such as defect management system (DMS) and version management system (VMS), are used for the management of defect reports and source code. Usually, these systems are freely available to users and developers. While they help with the management of the project, they also provide a large amount of the data that can be used in the analysis of the project and its processes. However, in spite of the management systems and project guidelines, development and maintenance processes are not well known.

Nowadays, processes are measured and improved to gain better quality and more efficiency. However, in this thesis the purpose is not to measure process from that viewpoint. The purpose is to describe and evaluate the maintenance process, and determine whether it is reliable from the viewpoint of users. The Software Engineering Institute (SEI) defines reliability as follows [SEI06]:

> "The capability of an implementation to maintain its level of performance under stated conditions for a stated period of time."

Generally, reliability, quality specifications and certificates rely on well-described and audited processes, while in OSSPs the processes and practices are poorly described and the boundaries between activities are unclear. For example, OSS is being developed and maintained simultaneously, but maintenance is rarely even mentioned.

The IEEE and ISO standards define maintenance as a post-delivery process whose purpose is to adapt the software to changed requirements or environments [IEE04]. In many OSSPs, the software has been delivered to the users and modifications are post-delivery activities. Since the users are also developers, they test software simultaneously. In this study, the approach is from the viewpoint of software maintenance instead of software development because the majority of users are not developers and the software has already been delivered.

Companies, governments and communities have got used to the fact that software manufacturers or service providers support software and guarantee required modification. OSSP usually does not have one particular actor who produces and maintains software. The majority of OSS developers and maintainers are volunteers and they may not be committed to maintenance. The continuity of maintenance relies on the number of developers and availability of the source code. To be sure that the needed changes will be made, one must make them oneself or be a friend of a developer who promises to make them. Otherwise, we can only hope that changes will be made.

This study researches the OSS maintenance processes. The purpose of this thesis is to analyse OSS maintenance processes and create a descriptive model. The model could be used to evaluate the reliability of the maintenance process in OSSPs. The objectives of this thesis are presented in Section 1.1. Section 1.2 outlines the structure of the thesis; the original publications are presented in Section 1.3.

1.1 Objectives

The purpose of this work was to create an approach and frameworks for the evaluation of maintenance processes in OSSPs. The objectives were to:

i. Establish a framework for the technical characteristics of Open Source Software projects.

ii. Establish a framework for the Open Source Software maintenance process.

iii. Create an evaluation framework and a tool that can be used to analyze Open Source Software maintenance processes.

iv. Validate the suitability of the evaluation framework and tool by analyzing Open Source Software maintenance processes through case studies.

Issues related to the framework for the basic characteristics of Open Source Software projects. OSS can be any kind of software and its target audience can vary from developers to users of desktop software [KoH04]. One or many leaders may lead OSSPs, or OSSPs can be bazaar-stylish [Ray99]. OSSPs or companies may offer user support and other services, or users may be left on their own [LeT00, LeT02, KoH04]. The purpose of the framework is to provide a general view of OSSPs.

Issues related to the framework for the Open Source Software maintenance process. Several standards and studies (e.g. IEEE standard for maintenance [IEE04], Software

Engineering: software life cycle processes [ISO02a, ISO02b], Software maintenance: concepts and practice [TaG03], IT Infrastructure management [OGC02a], IT application management [OGC02b]) describe the activities involved in the maintenance process, but they are not directly applicable in the OSS maintenance process. By mapping and comparing the maintenance activities of the OSS maintenance process to standards, it is possible to provide a framework that describes the general activities of the OSS maintenance process.

Issues related to the evaluation framework and tool that can be used to analyze Open Source Software maintenance processes. DMS and VMS are used to monitor defects and the source code of the software, but they also provide a large amount of data that can be used to analyze OSS maintenance processes [ADH04, Kop06a, Kop06b]. Therefore, we have created an evaluation framework for Open Source Software maintenance that has three aspects of the maintenance process of OSSPs: process activity, workflows and management. Because of the large amount of data, analyzing manually is almost impossible. Earlier studies have a few analysing systems for these DMSs and VMSs, such as SoftChange [Sof06], GlueTheos [Lib06] and CVSAnalY [CVA06]. However, none of them provides information for the evaluation of the maintenance. In addition, they analyse either VMS or DMS only. Therefore, we built a system that retrieves and analyses data from VMS and DMS.

Issues related to analyzing Open Source Software maintenance processes. Since we created an evaluation framework and automatic analysis system, we tested and validated them using seven OSSPs. In addition, analysis of these OSSPs provided information about the maintenance processes of OSSPs.

1.2 Organization of thesis

The organization of the thesis is presented in Figure 1.1. The figure shows the relations between the research objectives, topics of the thesis and original publications.

As we see from the figure, to provide an answer to the first (i) research objective Section 2 introduces the basic concepts of Open Source Software and the *evaluation framework for Open Source Software*. The section starts by describing the three major factors in OSS: licensing, community and services. Then it reviews the definition of the Open Source licenses and points out major differences between popular licenses. The section also briefly describes different organizational structures of the communities and the services provided.

The basic concepts and definitions of the software maintenance process are introduced in Section 3. We present an overview of the maintenance process described in the IEEE and ISO standards, and a proposal for the *Open Source Software maintenance process framework*, which provides answers to the second (ii) research objective.

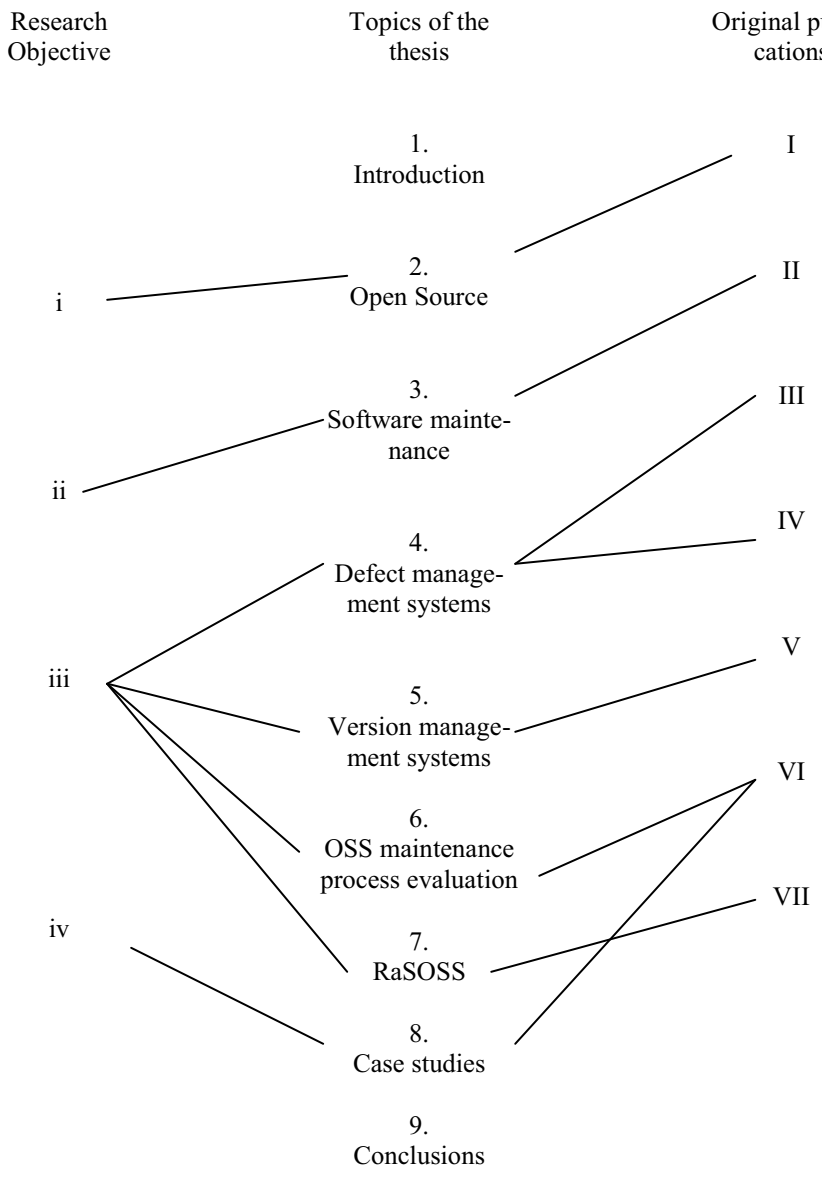


Figure 1.1. Research objectives, topics of the thesis and original publications.

To answer the third (iii) research objective, we present, in Sections 4 and 5, DMS, VMS, and their roles in the maintenance process. The sections start with an overview of the systems and their roles in the maintenance process. Then, they continue with a description of how the systems can be used as data sources in the evaluation of the maintenance process. We also introduce the most popular Open Source DMSs and VMSs and their purposes, usage and suitability for research. Then, in Section 6, we present the *evaluation framework for the Open Source Software maintenance process*. Section 7 presents the RaSOSS.

Finally, we evaluate the *evaluation framework for the Open Source Software maintenance process* with seven case studies, to obtain an answer for the fourth (iv) research objective. The results are presented in Section 8. Section 9 reviews the results of this thesis and introduces future work.

1.3 Original publications

This thesis is based on and extended from the author's contributions to the publications below. The author of this thesis is responsible for the contribution of papers IV, VI and VII. Joint work with the second author of Papers I, II and III helped to clarify the objectives of the thesis. In Paper V, the second author participated mainly in the writing of the research paper.

In Paper I, OSS is examined from the viewpoints of the product, project and services. The *evaluation framework for Open Source Software* was created and validated with eight OSS case studies: three operating systems, three server software, and two office software were evaluated. The results show that the framework was appropriate for the basic evaluation of OSS. It helped in classifying Open Source software products and projects. However, that approach does not include deeper analysis of the software or project processes.

Previously, the OSS maintenance process has been considered to be unorganized and unstable. Paper II presents the *Open Source Software maintenance process framework*. The study and the framework described show that the maintenance process of the OSSPs was similar to the ISO/IEC maintenance process model. The major difference was that the Open Source Software maintenance process did not have retirement activity, which existed in the ISO/IEC Maintenance process. Furthermore, the study show that the Defect management system (DMS) and Version management system (VMS) play essential roles in the OSS maintenance processes.

Papers I and II show that the DMS is used in OSSPs for management. Therefore, Paper III presents a study of the defects. It starts with two hypotheses. The first hypothesis was that the rate of enhancement requests is a metric for the maturity of the software. The second hypothesis was that the rate of useless defect reports is a metric for the quality of the defect reports. The paper presents two case studies, the Apache HTTP server and Mozilla Firefox browser, which were studied to test the hypotheses. Defects were gathered from DMSs. The study show that the rate of the enhancements was low in both

projects. Furthermore, the rate of defects reports which did not lead to any changes ranged from 50 to 70 percent.

When Paper III showed that the majority of defect reports do not lead to changes, we started to evaluate defect life cycles in Paper IV. The paper describes an approach for defect life cycle analysis. We analyzed two case studies to test the approach and found out that defect life cycles were much simpler than the *framework for Open Source Software maintenance process* suggests. The majority of the defect reports were just opened and resolved without any other steps in the life cycle.

When Paper III shows that only a small proportion of defect reports lead to changes, we started to study, in Paper V, the version management system more intensively. We created an approach that can be used to analyze the relationship between the changes in the source code and defect reports. The approach maps the changes in the source code to the defect reports if they are related. We evaluated our approach with two case studies, Apache and Mozilla, and found it suitable. The case studies show that the majority of the changes are not related to the defect reports.

When Papers IV and V showed that only a small proportion of defects led to changes and the majority of the changes were not due to the defect reports, we created a tool that could be used to evaluate more OSSPs. The tool, RaSOSS, implemented approaches that had been described in earlier papers. Paper VII is dedicated to the descriptions of the tool and its metrics. The paper presents the modular structure of the tool, database structures and the systems that are supported.

When we had finished RaSOSS, we selected seven case studies for testing. The *evaluation framework for Open Source Software maintenance* and results are presented in Paper VI. These cases reinforced the previous finding that defect life cycles have two or three steps. The majority of the defect reports were just opened and resolved without any other steps in the life cycle, which indicates inefficient usage of DMS. Furthermore, we analyzed the same case studies using the approach presented in Paper V and found out that in these cases, the majority of the changes in the source code were not related to the defect reports.

2 OPEN SOURCE

Nowadays, Open Source is more than a description of the practices that are used to develop products. The difference between the terms Open Source, Open Source Software and Open Source Software projects is the following: *Open Source* (OS) describes practices such as licensing and development models. *Open Source Software* (OSS) is a software which has been licensed with an Open Source license and has been developed using Open Source practices. Software development is nowadays considered to be a project. A project is defined in ISO 9000 as follows [ISO00]:

> "A project is a unique process consisting of a set of coordinated and controlled activities with start and finish dates, undertaken to achieve an objective conforming to specific requirements, including the constraints of time and resources."

An *Open Source Software project* (OSSP) is a temporary process to create unique software. Originally, the term Open Source refered to the users' right to get the source code of the software. The Open Source Initiative (OSI), which is a non-profit organization, launched the term in 1999 and its purpose is to promote OSS [OSI06]. The initiators of OSI were Bruce Perens and Eric Raymond. OSI has summed up the idea of OSS as follows [OSI06]

> "When programmers can read, redistribute, and modify the source code for a piece of software, the software evolves. People improve it, people adapt it, and people fix bugs. And this can happen at a speed that, if one is used to the slow pace of conventional software development, seems astonishing."

The term *Free Software* (FS) has a similar meaning as Open Source. The major differences between the terms are their issuers and political viewpoints. Richard Stallman issued FS in 1983 when he started the GNU project. The Free Software Definition [FSF06a] defines FS, and it has been published by the Free Software Foundation (FSF). The definition describes four rights [FSF06a]:

- Freedom 0: "The freedom to run the program, for any purpose."
- Freedom 1: "The freedom to study how the program works, and adapt it to your needs. Access to the source code is a precondition for this.
- Freedom 2: "The freedom to redistribute copies so you can help your neighbour."
- Freedom 3: "The freedom to improve the program, and release your improvements to the public, so that the whole community benefits. Access to the source code is a precondition for this."

Although it mentions the rights stated in the OSI, the definition does not claim that the software should be free of charge. However, this term did not convince companies and

other parties, and they launched the OSI in 1999. *Libre Software* is a synonym for FS, which emphasizes freedom.

To understand OSSPs practices, three major issues need to be explained: licensing models, community and development models, and services. Software is always a combination of these issues for the user. A license describes explicitly the users' rights and obligations. Open Source licensing models are described in Section 2.1. A community is an actor that users deal with, and its organizational structure affects the development models of the software. The community and development models are presented in Section 2.2. The community or companies [DOS99, Ros00] may provide services such as maintenance and support. Service types and their providers and importance are presented in Section 2.3 [DOS99, Ros00]. The *framework for Open Source Software* is presented in Section 2.4, but the original framework is presented in Paper I.

2.1 Licensing

The software is protected by copyrights which protect the results of creative work. The copyright gives (to the creator) exclusive right to distribute and copy the work. In Finland, copyright is defined in the Copyright Law (Tekijänoikeuslaki) [Fin06] and it is recognized internationally [Ros04].

Open Source licenses give users the right to redistribute and modify software, which was previously the right of the copyright holder only. The fundamental requirements for Open Source licenses are defined in the Open Source Definition released by the Open Source Initiative [OSI06]. The definition has ten clauses [OSI06]:

1. Free Redistribution
2. Source Code
3. Derived Works
4. Integrity of The Author's Source Code
5. No Discrimination Against Persons or Groups
6. No Discrimination Against Fields of Endeavour
7. Distribution of License
8. License Must Not Be Specific to a Product
9. License Must Not Restrict Other Software
10. License Must Be Technology-Neutral

These clauses give users the freedom to use, modify and distribute the software. They also eliminate the possibility of artificial limitations, which are against the spirit of freedom [OSI06]. Although all Open Source licenses provide these rights, there are also differences between licenses. The major differences are in the implementation of copyleft, which allows the user to freely copy and modify source code. It also allows the redistribution of software for a price or for free as long as the source code is licensed with the same license. Open Source licenses can be classified in terms of the implementation of copyleft (presented in Table 2.1). [Ins06]

Table 2.1. Characteristics of classified licenses.

Characteristic	Licenses without copyleft effect	Licenses with strong copyleft effect	Licenses with restricted copyleft effect	Licenses with limited freedom of choice	Licenses with privileges
Derivate work can be released with a different license	Yes	No	Yes[3]	Depends	Yes[1]
Source code can be combined with software that has been licensed with a different license	Yes	No	Yes[2]	Depends	
Binary software can be released without the source code	Yes	No	No	Depends	Yes[1]
Viral	No	Yes	Yes	Depends	Yes
Persistent	No	Yes	Yes	Depends	Yes

1) If defined in the special privileges
2) With technical limitations
3) With a limited set of licenses

Licenses without copyleft effect give all the rights and privileges from the Open Source Definition. These licenses also give the right to re-license a program with another license, if the work is modified. These licenses are meant to spread new technology and distribute the software. Examples of these licenses are Apache, Berkeley Software Distribution (BSD), Massachusetts Institute of Technology (MIT), and X licenses.

Licenses with strong copyleft effect are viral and persistent. Derived work must be distributed with the same license when the license is persistent, and combined work must be licensed with the same license if it is viral. A viral license is the 'virus' that infects all combined work. It might not be possible to combine source code from different software if they have been licensed with different licenses that have viral effects. Examples of these licenses are the Gnu Public License (GPL) and IBM Public License.

Licenses with restricted copyleft effect share similarities with *licenses with strong copyleft effect*, but they allow changing of the license with some restrictions, for example license change is allowed from the Lesser Gnu Public License (LGPL) to the GPL. Because the viral effect is limited, it is possible to create software libraries that can be used as a part of other software. Examples of these licenses are the Lesser Gnu Public License (LGPL) and Mozilla Public License (MPL).

Licenses with limited freedom of choice may limit the distribution method of the derived work. The license may require that the original name of the software is not used with the

modified software, or that derived work is distributed as patch files. Examples of these licenses are the Artistic License and Latex Project Public License.

Licenses with privileges include special privileges for a company or foundation. These privileges might be the right to redistribute executable software with a proprietary license. Examples of these licenses are the Apple Public Source License (APSL) and Sun Industry Standards Source License (SISSL) [Ins06].

Some licenses, such as those with strong copyright effect, limit the usage of the software and it may not be a part of any other software which is not licensed with same license. Sometimes this may become a problem.

For example, let us assume that software A has been licensed with the GPL and software B has been licensed with the Apache license. Parts of software A cannot be included in software B because the GPL prohibits changing the license. However, controversially, any part of software B can be included in the software because the Apache license allows re-licensing. If the modified parts of the new combined work, they cannot be returned to the software A due different licenses.

Another licensing issue may become a problem if the software has been licensed with a license that includes special privileges. For example, this may happen if software A has been licensed with the SISSL and some developers are concerned that Sun Microsystems may distribute (sell) software without the source code, and retire from the project.

In addition, too liberal licensing may become a problem. If the license gives too many rights, the project may fork into multiple projects which are competitors to each other. The developers will be divided between the forked projects, which leaves fewer developers for each project.

2.2 Community and development models

The users and developers of software form a community which always has an organizational structure with roles. The members of the community can be categorized according to their roles as users, developers, core developers and associates [NYN02]. The users use the software and they may give feedback. The developers use and develop the software. The core developers are the initiators of the development or play a distinguished part in the development. The associates are foundations or companies that support development by donating resources such as hardware or knowledge, promoting the software to potential users and developers, or by giving financial support. Furthermore, the associates can assign their employees to participate in the development[San01].

The organizational structure of the community is used for monitoring people and to share tasks. OSSPs have two basic structures for the community: hierarchical and network. A community with a hierarchical structure (Figure 2.1) has a definite leader or leaders who control the development of the software. A community with a network

structure (Figure 2.2) has distributed leadership and therefore it is more like a peer group[San01].

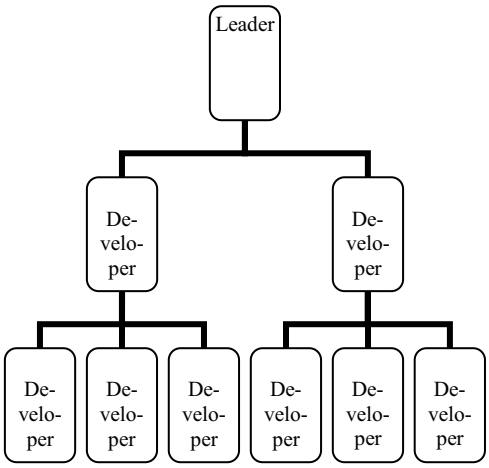


Figure 2.1. Community with a hierarchical structure.

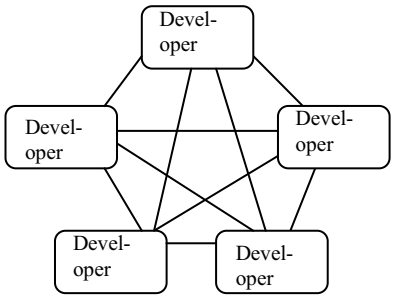


Figure 2.2. Community with a network structure.

In a hierarchical community, the leaders, who may be persons, communities or companies, make decisions and plan the future. The users and associates follow instructions from the leaders. To earn a higher position in the hierarchy participants have to contribute significantly to the project, because positions are earned with respect and trust [San01].

In a community with a network structure, the positions and influence of the participants are related to social contacts with the other members. The absence of leaders causes more inconsistent and inaccurate planning and management than in a community with a hierarchical model [San01].

In addition to community models, software always requires a development model. Eric Raymond in *The Cathedral and the Bazaar* [Ray99] presents two development models, a cathedral and a bazaar. These models have been foundational to research of Open Source development models. They are theoretical but their characteristics can be recognized in the communities. In the *cathedral model*, developers will implement software from the designs of the leaders. Therefore, the cathedral model is considered to be similar to the approach of software engineering. In the *bazaar model*, the development and design are open and everyone is allowed to participate by creating modifications. In the bazaar model, there are no exact plans or design for the software.

However, Raymond [Ray99] did not present the roles that were associated with the models. Rothfuss has studied the development of OSS and defined several roles in the development process [Rot02]. He describes two actor groups, the users and contributors. He also argues that there are two types of contributors, insiders and learners. The learners do not have enough knowledge or skills to make significant contributions but they can help

in minor things. The contributors are actual developers in the project. In addition to actors, he defined several roles for each actor type and an overview of the process. However, we would add a third actor type: an associate, as suggested by Nakakoji et al. [NYN02].

Capiluppi, Lago and Morisio [CLM02] have presented general characteristics of the Open Source process. The management processes of OSSPs have been studied more carefully. Erenkranz [Ere03] presented a process for release management in the Linux kernel and Apache development. Asklund and Bendix have presented a model for the configuration process in OSSPs [AsB02].

Since OSSPs use a distributed approach to development despite the organizational structure, they need channels to communicate and coordinate work. The communication channels can be email, mailing lists, newsgroups, bulletin boards, instant messengers or chat forums. However, these channels may be hard to manage if there are very many defects. OSSPs use DMSs to manage defects and enhancements requests. They also use VMS to manage the source code of the project. DMS and its basic concepts will be presented in more detail in Section 4. VMS and its basic concepts will be presented in more detail in Section 5

2.3 Services

The usage of OSS is usually an interaction between these three actor groups: users, developers and associate companies. While proprietary software companies usually provide services to support their software products, the developers of OSS do not provide similar services. Regardless of the type of software, users require different kinds of services, such as consulting and training, and there are vendors who provide the missing services [LeT00, LeT02]. The availability of the services depends on the software developers and associates.

The most important services are maintenance, customization, and consulting and training. Usually, software is built to meet quite general user needs. However, some users may require features that are not so general. The developers may not be willing to implement these specialities and then the user needs a service that produces a custom version of the software, which is the result of customization. The community rarely provides customization service.

Consulting services, such as support services, and training services are usually not provided by the community. The developers may provide some support in mailing lists and bulletin boards. However, the level of these services can vary from nothing to the very professional. If there are a large number of active users and developers, this service is more likely to be available, but there is no guarantee of that. Therefore, associates who have commercial purposes provide the consulting and training services

Usually, the community maintains the software but it may not guarantee the continuity of maintenance. Since some users require and are ready to purchase maintenance and support services, this creates an opportunity for companies to offer such services.

2.4 Evaluation framework for Open Source Software

The development processes have been studied through the roles, development and licensing models, economics and release management [Rot02, Ros04, LeT00, LeT02, MFH02, NYN02, Gia05, HNH03, and Lon05]. Since the characteristics of OSSPs have been presented in earlier studies, we created a framework using those characteristics. The evaluation framework for Open Source Software, presented in Paper I, can be used to create a general view of a project.

The framework gathers information from the viewpoint of the information product, project and services. The information product viewpoint observes the type of software, the scale of the user group and the licensing model. The project viewpoint gathers information about the organization of the community and management systems such as DMS and VMS. The service viewpoint gathers information about what services are provided, and by whom they are provided. An outline of the framework is presented in Table 2.2.

Table 2.2. Outline of the evaluation framework for Open Source Software.

Viewpoint/attribute	**Type**
Information Product	
Software name	String
Software type	String
Available since	Month and year
Number of available languages	Number
Number of users	Estimate number
License and License group	Name and group
Copyright holder	Name of company, foundation or person
Project	
Project leader	Name of company, foundation or person
Associate	Name of company, foundation or person
Intended audience	String
Development model	Cathedral, bazaar or other
Public DMS	No/Yes and type
Public VMS	No/Yes and type
Services	
Software customization	No/Yes and provider
Consulting and training services	No/Yes and provider
Maintenance	No/Yes and provider

As we can see from the table, each viewpoint is divided into several attributes. The attributes of the information product viewpoint collect general information about the

software and immaterial rights. They can be used when searching for candidates for the software. The software type, availability and license are general attributes that cannot be ignored. The number of users describes how widely the software is used. The purpose of the project viewpoint is to provide information about the project itself and its environment. It can be used to evaluate the general structure of the project. The project leader and development model provide a brief overview of the style of management in the project. The availability of DMS and VMS provide the users with a channel to participate in and they provide more information for researchers and acquirers about management and processes. In addition, if the project has many plausible associates, it may be more trustworthy. The availability of services is very important for some users and acquirers.

As our first (i) research objective was to provide a framework for the evaluation of OSS from a general viewpoint, this framework fulfils that objective. However, the problem with the framework is that it is quite superficial from many perspectives and so it is suitable only for searching for candidates for further analysis. It leaves several open questions, such as maintenance. If maintenance is not provided by the commercial vendor, can it be trusted? If maintenance is provided by the community or developers, can it be trusted?

Some answers to those questions can be provided by studying DMSs and VMSs. These systems provide information that can be analyzed and used to answer those questions. The simplest approach to analyze and evaluate the maintenance process from the Open Source process is to use publicly available data from their DMS and VMS if they are used in the project.

3 SOFTWARE MAINTENANCE

In a changing environment, the software is also supposed to change. Software maintenance is one of the processes in the software life cycle. Its purpose is to keep software operational, to prevent and correct faults in the software, and enhance the functionality of the software [Ben00, IEE04, ISO02a, ISO02b]. According to the Software Body of Knowledge (SWEBOK) [IEE04], software maintenance is defined in the IEEE Standard for Software Maintenance as follows [IEE04, p. 6-1]:

> "Modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment."

Over the years, several maintenance models have been proposed, each one emphasizing a particular aspect of software maintenance. However, these models share some common activities. Before the arrival of actual maintenance process models, several models defined software change activities and tasks. The majority of the models had the following three activities: understanding software, implementing change within an existing system, and retesting the modified system.

To define a software maintenance methodology or processes, it is useful to take into account international standards. IEEE 1219 [IEE98], ISO/IEC 14764 [ISO99] and ITIL [OGC02a] are three internationally regonized standards that define maintenance process models [KaM06]. These standards specify the activities needed and their inputs and outputs [IEE04, p. 6-6] but they do not provide step-by-step instructions for maintainers, and this lack sometimes make maintainers' work more difficult. Although it is difficult to provide guidelines with universal validity, the standards expose the necessity to execute some activities and tasks.

A process model at a higher abstraction level is required to understand the relationship between the users and software change. The processes are always initiated by an input. In the maintenance process, an input is needed for a change such as a defect, bug or problem report, or enhancement request. These reports and requests are used to describe software functionality that is not fulfilling requirements, or some needed improvements. The defect is defined in the ISO 9000 standard as [ISO00]:

> "The non-fulfilment of intended usage requirements. The departure or absence of one or more quality characteristics from intended usage requirements. It is also a lack of quality in a product in relation to its use (minor if this does not hamper usage, major if it prevents one from using it, and critical if it prevents a broader entity from functioning) resulting from deviating from the specification or a faulty specification."

We use, in this thesis, the term defect because it is used in ISO 9000 and it may have the widest meaning. Because of variations in the term, DMSs are also called bug manage-

ment and bug tracking and problem management systems in some contexts. These terms have similar meanings but they emphasize different aspects or uses.

The software maintenance process starts with the input defect report. Thereafter, the defect report may generate a change request, which is an input for the software change activities. These activities produce a new modified version of the software for the users. Figure 3.1 presents the context of the maintenance process.

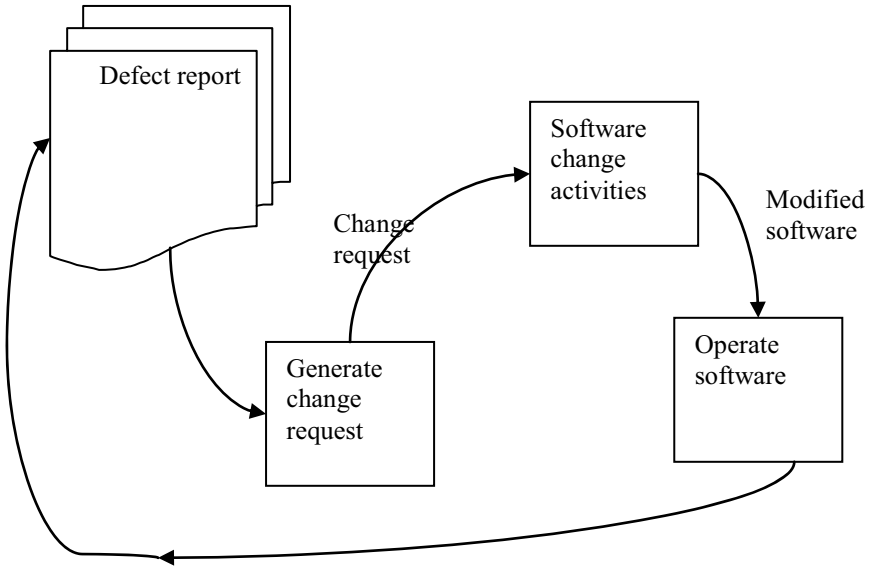


Figure 3.1. Context of the software maintenance process.

While the context diagram gives an overview of the software maintenance process, it does not define the tasks or even the activities to be executed. As we stated earlier, the activities and tasks of the maintenance process are described in several standards such as ISO/IEC 12207, IT Infrastructure Library (ITIL) and ISO/IEC 14764. ISO/IEC 12207 describes all the processes that exist in the software life cycle and therefore it is quite general. The ISO/IEC 14764 standard elaborates a description of the maintenance process of the ISO/IEC 12207 standard [IEE04].

The IEEE 1219 and ITIL description are separate from the ISO/IEC standards, although the activities of the IEEE and ISO/IEC standards are similar [IEE04]. ITIL describes the maintenance process a bit differently, because it describes it from the viewpoint of the infrastructure and service management [OGC02a, OGC02b]. Takang and Grubb [TaG03] have developed earlier maintenance process models that were the basis of the development of standard models.

These process models are quite general but to provide a basis for understanding the OSS maintenence process we present, in Section 3.1, the software maintenance process that is

described in the ISO/IEC 12207 and ISO/IEC 14764 standards. Then in Section 3.2, we present briefly the Open Source Software maintenance process framework. The ISO/IEC and OSS maintenance processes have been compared in Paper II. Furthermore, Paper II presents the framework and evaluates case studies.

3.1 ISO/IEC Software Maintenance Process

An overview of the ISO/IEC maintenance process is presented in Figure 3.2. As the figure shows, the ISO/IEC maintenance process has been divided into six activities. The activity Process Implementation is considered to be only a pre-delivery activity in the process. Pre-delivery activities are performed in the planning or development state of the software life cycle. The purpose of pre-delivery activities is to create an environment and plans for the maintenance process. Other activities are considered post-delivery activities and are performed after product delivery. Their purpose is to plan and implement modifications in a controlled way. Furthermore, they support software migration and retirement. [IEE04, ISO02a, ISO02b]

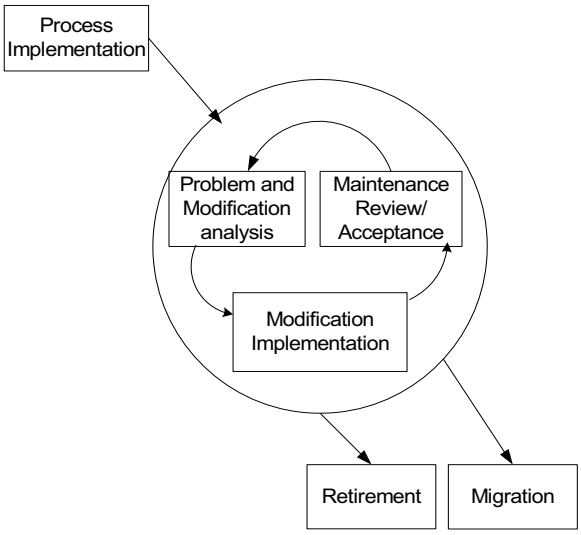


Figure 3.2. ISO/IEC 14764 maintenance process. [IEE04, p. 6-7]

The activity process implementation includes the following tasks: developing maintenance plans and procedures, establishing procedures for modification requests, and implementing the configuration management process. The purpose of process implementation is to form an environment for the effective management of the maintenance process.

The problem and modification analysis is a post-delivery activity. Its purpose is to provide procedures for the initial analysis of the problem and modification, problem verification, and documenting that the problem or modification request is stored in an appro-

priate way. Another purpose is to develop initial options for implementation and obtain approval for the modification.

After approval, the process continues with the activity modification implementation. Its purpose is to analyze the problem or modification with more details and develop it. The modification implementation activity includes the testing of the modification.

The maintenance review and acceptance activities are a part of the quality control. Their purpose is to ensure that the modification fulfils the requirements of quality management.

The purpose of migration is to ensure a smooth transition to the corrected or enhanced software. This activity includes the development of plans, user notification, migration and reviews. The actions in migration should be planned in a way that minimizes the disadvantages of migration. The activity should ensure that the data and system are available after migration.

Retirement ensures that the information will be available after the software is replaced with another one. Therefore, the retirement activity describes the replacement of the system. It should include procedures that ensure the availability of the data after the replacement.

3.2 Open Source Software maintenance process

When we analyzed the OSS maintenance process in Paper II, we found it quite similar to the ISO/IEC maintenance process model. The most visible difference was that the Open Source Maintenance process does not have well-described migration and retirement activities, and planning tasks are also not so disciplined. The OSS developers and users are distributed geographically and therefore management of the defect reports and source code is an important factor for the success of the project. Therefore, OSSPs use, at least, two kinds of management systems: a defect management system (DMS) and a version management system (VMS). The purpose of a DMS is to establish procedures for managing defect reports. DMSs give procedures for reporting and assigning the defect reports and modification requests. Furthermore, DMSs provide flexible possibilities for tracking and controlling defects because they store all the changes made in defect reports. VMSs establish an environment for source code management. Figure 3.3 presents the activities of the OSS maintenance process.

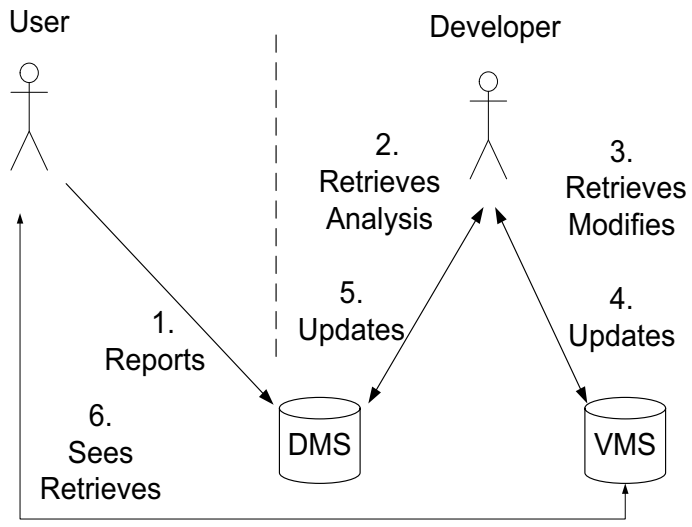


Figure 3.3. Activities of the OSS maintenance process.

The activities involve the following:

1. Users may find a defect and report it to the DMS. The users may also check that similar defects that have not yet been reported.

2. Developers retrieve defect information from the DMS, ensure the existence of the defect, analyse it and complete missing attributes of the defect.

3. In the implementation phase, the developer checks out (=retrieves) the source code from the VMS and modifies the source code to fix a defect or implement an enhancement. Furthermore, the developer submits the modification to the VMS.

4 . The modification is reviewed before it is accepted and merged to the main version of the source code in the VMS.

5. The status and resolution of the defect is updated into the DMS.

6. Later, a new version of the software is available from the VMS or from the project web site. Then the users can retrieve the modified version.

After the maintenance activities, the DMS and VMS should have similar information about the resolution and status of the defect. The defects are resolved during the main-

tenance process but there are several possible resolutions because not all defects lead to modifications of the source code.

3.3 Differences between maintenance processes

Maintenance processes have been described in several standards but none of them was directly applicable to the OSS maintenance process. The framework presented in Paper II showed that the DMS and VMS play significant roles in the OSS maintenance process. Comparison of the standards and the Open Source Software maintenance process framework showed that the processes are quite similar but there are also significant differences. The major differences were a lack of the retirement activity, and obscure migration activity. Another difference between the ISO/IEC 14764 maintenance process and the OSS maintenance process is in modification review and acceptance [AsB02].

Modifications are accepted before implementation in the ISO/IEE maintenance process but after implementation in the OSS maintenance process [AsB02]. Although this may seem irrelevant, it has a very significant effect on the maintenance process. In ISO/IEC 14764 maintenance, the process resources can be allocated effectively but the OSS developers assign their work independently. In addition, in the ISO/IEC 14764 maintenance process changes are accepted before implementation, but in the OSS maintenance process but they are not specifically assigned to anyone. The developers can retrieve any of the defects and start to implement the required modifications. After implementations, the modification is reviewed and can still be rejected. Rejection of the modifications means that the work that has already been done was wasted.

4 DEFECT MANAGEMENT SYSTEMS

In software development, defect reports and enhancement requests can easily become unmanageable. When the flow of reported defects, requested enhancements and support requests increases, it is impossible to manage all of them with email and post-it stickers. Therefore, many companies and OSSPs have built systems for managing these reports and requests.

Management systems store requests and reports in databases where they can be retrieved, assigned, processed and closed. The basic function of these management systems is to create a ticket which describes the related request or report, work in progress and all related communications. These management systems are also referred to as ticket-tracking systems [OCG02a]. They are widely used in helpdesks and their customer support services.

Ticket-tracking systems are provided by for example IBM, CA, and Remedy. However, due to their price and the ideology behind Open Source, OSSPs have created their own ticket-tracking systems. However, these OSS systems are not called ticket-tracking systems although their functionality and purpose are the same. OSSPs use many names for their systems, such as bug-tracking systems, defect management systems, and issue management systems. All these terms have a similar meaning, although they may emphasize different aspects. For example, bug-tracking systems are intended for bugs only, while an issue-tracking system may also handle other issues such as support requests. In this thesis, we use the term defect management system (DMS).

Maintenance standards (e.g. ISO/IEC 14764) classifies maintenance into four categories: preventive, corrective, perfective and adaptive. Furthermore, it refers to preventive and corrective maintenance as corrections, and perfective and adaptive maintenance as enhancements [IEE04, p. 6-3]. Nevertheless, other classifications are also used [KaM06]. In OSSPs, classification is simpler and has two categories: defects and enhancement requests. Defects include bugs, errors, faults and failures, even if they are separated in other contexts.

Next, Section 4.1 overviews the relationship between a DMS and the maintenance process; Section 4.2 describes the use of a DMS as a data source; and Section 4.3 presents a few example DMSs with a brief comparison.

4.1 Relationship between a DMS and the maintenance process

As we saw in Section 3, DMSs provide procedures for defect reporting and management. The reported defects are stored as defect reports in the database of the DMS. The users are able to browse and search defect reports; they are also able to retrieve summaries.

The developers can modify the stored information about the defect reports, assign defects and close defects. The common features of DMSs are presented in Table 4.1.

Table 4.1. Features of a DMS.

Feature	Explanation
Defect reporting	Users and developers can submit defect reports to a DMS.
Storing data of defects	All essential attributes presented in Section 4.2 are stored in the system.
Defect status management	Developers can change the status of the defect, so users and developers know where the defect is going.
Assignment of defects	It must be possible to assign a defect to a single developer or group of developers, so that overlapping work can be avoided.
Defect resolution management	It must be possible to assign a descriptive resolution for defects, so users and developers know how the defect was resolved.

In many cases, defects can be classified into three groups according to the type of resolution. Therefore, we grouped defects according to results into three groups: not resolved, fixed and non-fix-inducing. The first group contains all defects that do not have a resolution. The second and third groups contain defects that are already resolved and have caused changes. More specifically, the second group contains defects whose resolution is fixed. The third group contains defects whose status is also resolved but their resolution other than fixed. These three groups are presented in Table 4.2.

Table 4.2. Outcomes of defects.

Group	Resolution	Explanation
Not resolved	(empty)	Defect does not yet have a resolution.
Fix-inducing	Fixed	Defect is fixed and changes to source code are imported
Non-fix-inducing	Works for me	Defect does not occur with other users
	Won't fix	Defect is not a fault or a problem or it is a feature
	Invalid	Defect report is invalid
	Duplicate	Defect report duplicates another
	Other	Other non-described resolution

The classification of defects by outcome is quite similar in OSSPs. Figure 4.1 presents a classification of the defects according to the resolution and related activities of the maintenance process.

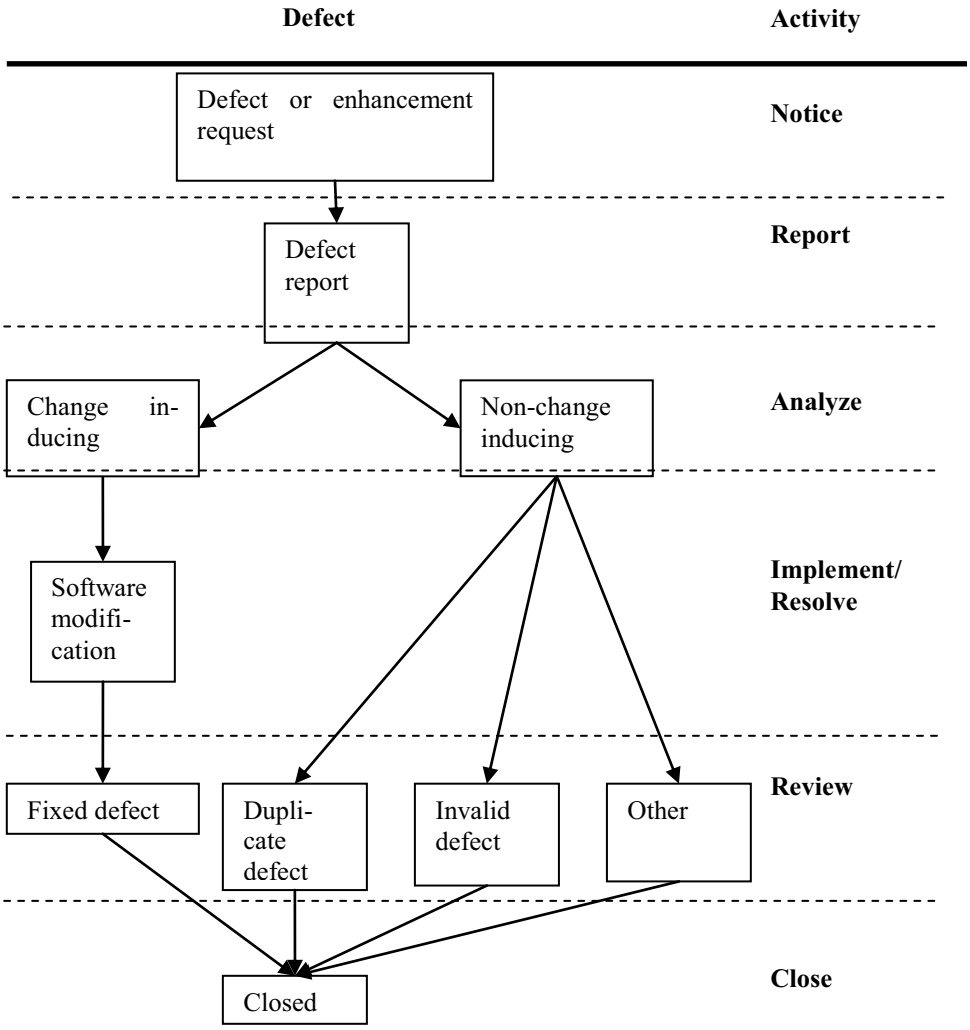


Figure 4.1. Classification of Defects according to resolution.

As the figure shows, a well-formed and accurate defect report could also be considered to be a modification request. When a defect induces changes to the software or the defect is resolved with reconfiguration of the users' system, the defect becomes resolved and fixed. When a defect is considered to be a copy, or in other ways describes an earlier reported defect, the defect becomes resolved and duplicate. When a defect report is badly formed, information is missing or it does not describe an actual defect, it is resolved and invalid. A software modification changes the source code of the software, which changes software behaviour. Furthermore, other resolutions, such as later and works for me, are possible but usually they do not lead to software modifications.

Not all of these defect reports report an actual bug. Therefore, it is useful to have a classification for different types of defects. Actually, some systems such as Tracker [Tra06] and Issue Tracker [OOo06e] use the term *issue* instead of defect. In their classification, a defect is one subtype of issues and there are other subtypes such as enhancement and support request. However, most DMSs do not have a specified attribute for defect types. For example, Bugzilla [Moz05] uses the attribute *severity*, which can have the following values: enhancement, and bug with several subtypes. Issue Tracker has an attribute for types that can have the following values: enhancement, bug, support and patch. Using these systems as a source, we created a generalized classification as shown in Table 4.3. However, most of the projects that were studied used only defects and enhancements in the classification.

Table 4.3. Classification of defects.

Type	Subtypes	Explanation
Enhancement		Request for new feature
Defect (bug)	Blocker	Blocks next release
	Critical	Blocks use
	Major	Prevents use of some essential functionality
	Normal	Software is still functional with all major functionalities
	Minor	Does not prevent use
	Trivial	Cosmetic or other user-interface problem
Support request		Similar to support service such as help desk
Patch		Users can submit improvements via DMS

4.2 A DMS as a data source

Since DMSs store a large amount of data about the maintenance process, they could be used for the analysis of the defects and process. The DMSs that were used in the case studies used a similar set of attributes to describe a defect and its status. These attributes are presented in Table 4.4.

Table 4.4. Attributes of the defect report.

Attribute	Usage
Id	Identification number
Environment	Identifies software and its environment where defect occurs such as product, component, version and platform
Status	Current status of the defect, such as resolved or new
Resolution	How was the defect resolved - did it induce fix or not?
Assigned to	Who is resolving the defect?
Severity	How significant is the defect?
Reporter	Who reported the defect?
Summary	Description of the defect
Type/Classification	Bug, enhancement, etc.
Activity log	What changes have been made to the defect reports?

In the defect analysis, we focus on statuses, resolutions and number of defects. When analysing the process itself we use an activity log to model workflows. But let us start with the method of the process analysis. An event-based process discovery framework was presented in 1995 by Cook and Wolf [CoW95]. The framework is based on a view where processes are considered to be a sequence of actions separated by events. The activities are performed by agents. The agents can be any kind of actors such as systems or persons. In the event-based process model, events are used to characterize behaviour. The framework was used to study bug repair in OSSPs models in 2003 by Ripoche and Gasser [RiG03a, RiG03b]. When an OSSP uses a DMS for defect management, the DMS produces an activity log that describes the changes that are made to the attributes of the defect reports. An example of an activity log is shown in Figure 4.3.

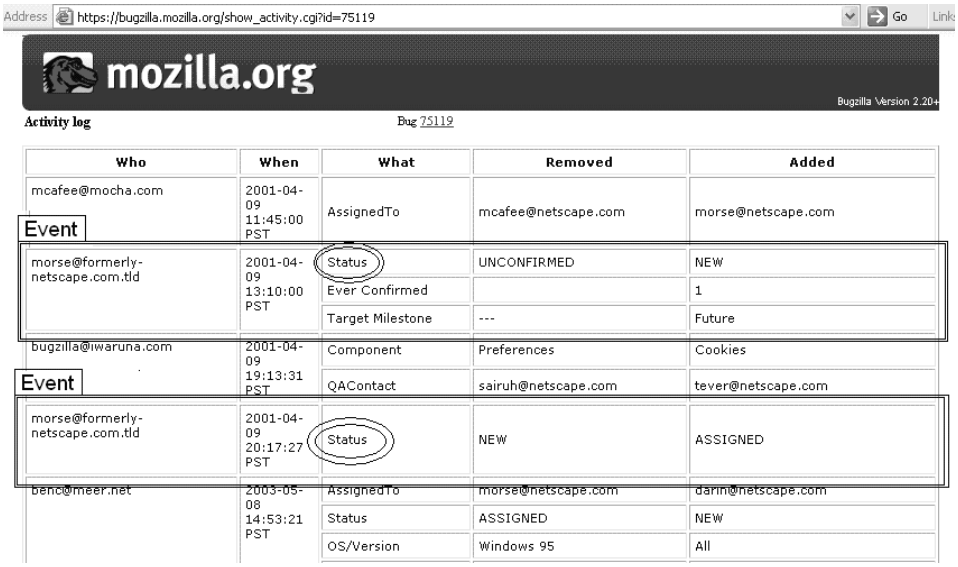


Figure 4.3. Activity log of a defect in Bugzilla. [Moz06a]

The activity log of Bugzilla stores the author (`Who`), time (`When`), attribute name (`What`), and old (`Removed`) and new value (`Added`) of each activity as an event. The activity log allows users and developers to trace the activities that have been done to the defect reports. In addition, the activity log makes it possible to analyse and model processes when all necessary events are logged. As we can see from the figure, one event can have one or more attribute changes. However, in this case, an event is an entry of the activity log that includes change of attribute status. So the event ends the previous activity of the process and starts the next activity.

Activities and statuses are presented in Table 4.5. The statuses shown in the table are omitted from Bugzilla but other DMSs use a similar set of statuses; however, they might also have some additional statuses such as need info and started.

Table 4.5. Activities and statuses.

Activity	Status
Report	Unconfirmed
Ensure existence	New
Analyze	Assign
Implement/Modify	Resolved
Review	Verified
Close report	Closed

As the table shows, each activity should change the status and therefore produce an event. Since events are extractable from the activity log, it is possible to model the workflow by chaining the events of each defect report. However, DMSs make it possible to transit from any of the statuses to resolved or closed, and then it is not always possible to establish a real workflow for each defect. For example, a defect report that is invalid or duplicate can transit directly to the status resolved from the statuses unconfirmed and new. However, it is not always possible that a defect follows the described workflow. Therefore, we model defect life cycles by using an event-based process discovery framework.

4.3 Example systems

There are many systems for OSS defect management, such as:

- Ticketing systems:
  - OTRS - Open Ticket Request System [OTR06]
  - Request Tracker [Bes06]
  - IssueTrackerProduct [ISS06]
  - Argus [ARG06]
- Defect management systems:
  - Bugzilla [Moz06a]
  - Tracker [Tra06]
  - IssueZilla/ IssueTracker [OOo06e]
  - Scarab [Tig06a]
  - Mantis [Man06]
  - Eventum [Eve06]
  - Gnats [FSF06b]
  - KDEBugTracking system [KDE06]
  - PhpBugTracker [PHP06]

Next, we briefly present five DMSs (Bugzilla [Moz05], Issue Tracker [OOe06], KDE Bug tracking system [KDE06], Eventum [Eve06] and Tracker [PHP06] that were used in the case studies.

Bugzilla is a well-known and widely used DMS in OSSPs. It was released in 1998 at the same time as Mozilla, which is one of the OSS pioneers. Its primary purpose was to manage bug reports in the Mozilla project. However, it was also adopted rapidly by other

projects. Bugzilla is a server based software and used through a web user interface. It stores all the attributes that were presented in Table 4.4 and many other attributes. Figure 4.4 shows a general view of a defect report in Bugzilla.

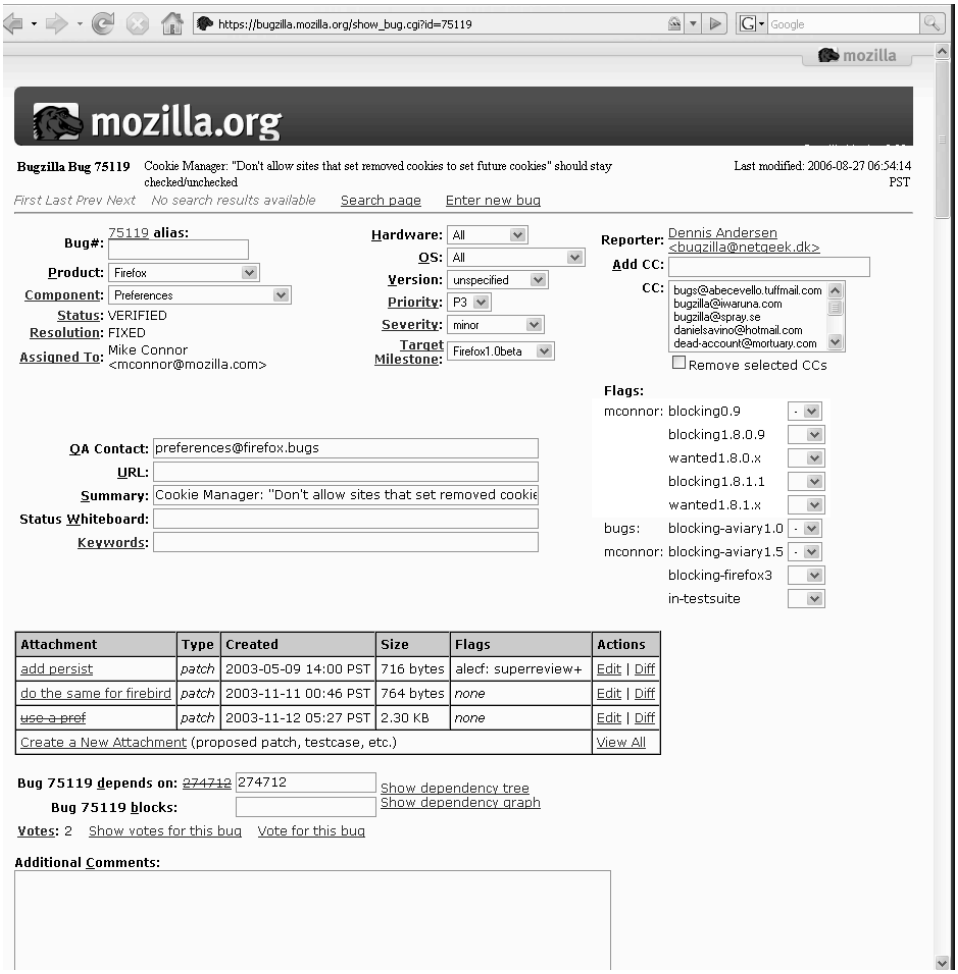


Figure 4.4. Defect report in Bugzilla. [Moz05]

Bugzilla has most of the attributes from Table 4.4. It does not have an attribute for the classification of defect types such as support request or feature request. The attribute severity is used for separating enhancements from requests. When enhancement requests are represented as defects, users may be confused.

Although Bugzilla has become a commonly used DMS, it has disadvantages. As we saw earlier, Bugzilla was created to manage only bugs, which is its major disadvantage. Other requests, such as support requests, were not manageable with Bugzilla. Therefore, several other DMSs have been developed and some of the other DMSs, such as Issue

Tracker and KDE Bug tracking system, are derived from Bugzilla. Issue Tracker was formerly known as IssueZilla and it is used in the OpenOffice.org projects. The KDE Bug tracking system is used in KDE projects. Since they are derivatives of Bugzilla, they store a quite similar set of attributes. Figure 4.5 and 4.6 show defect reports in the Issue Tracker and KDE Bug tracking system.

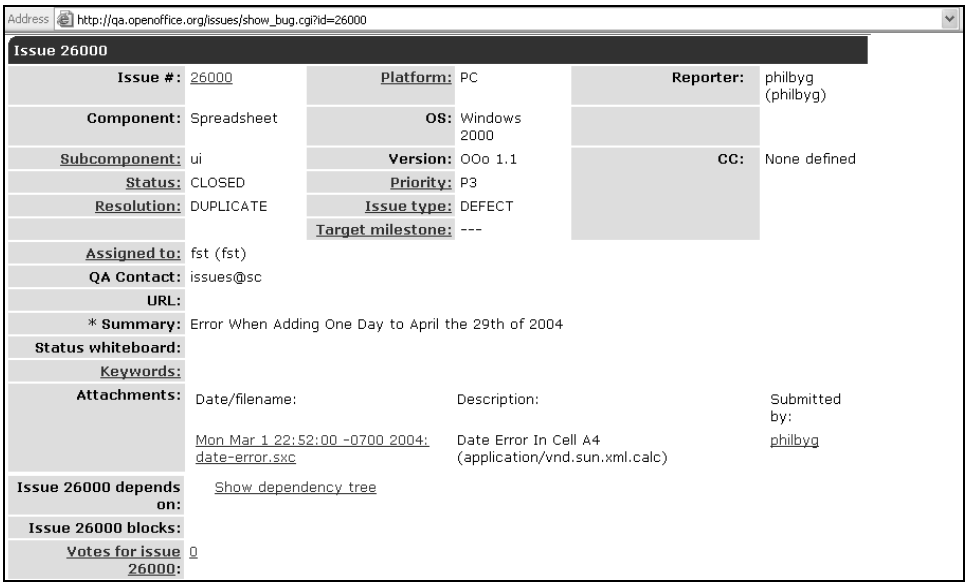


Figure 4.5. Defect report in Issue Tracker. [OOo06e]

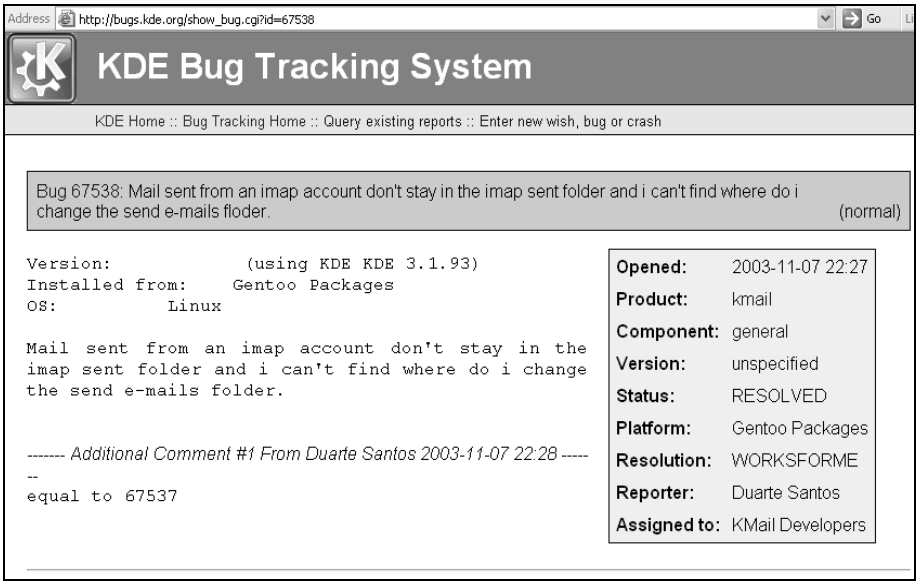


Figure 4.6. Defect report in the KDE Bug tracking system. [KDE06]

The figures show that both systems have most of the attributes described in Table 4.4. The layout and user-interface of Issue Tracker are similar to those of Bugzilla. However, the layout and user-interface of the KDE Bug tracking system looks more simplified than the others, because it seems to store less information. However, actually almost the same information is stored but only a part is shown. The most important improvement in Issue Tracker is the ability to manage support requests and other kinds of activities. The severity attribute in Bugzilla has been replaced with the attribute *issue type,* which describes the type of issue, for example a defect, enhancement request or support request. In the KDE Bug tracking system, the improvements are mainly in the user interface.

However, even though many DMSs are derived from Bugzilla, not all of them have based their systems on Bugzilla. For example, MySQL have developed their own DMS, Eventum [Eve06], which is used to manage defects in their own products. Another example is Tracker [Tra06], which is a part of the VA Software's SourceForge product. The SourceForge product is an environment for collaboration and distributed software development [VAS06]. Example defect reports in these systems are shown in Figures 4.7 and 4.8.

Bug #19936	DataReader already open exception		
Submitted:	19 May 12:15	Modified:	22 May 23:15
From:	Luke Quinane	Assigned to:	Tonci Grgin
Status:	Closed	Email Updates:	Subscribe
Severity:	S3 (Non-critical)	Lead:	Bugs System
Category:	Connector/Net		
Version:	1.0.7	OS:	Windows XP
Changeset:			

View | Add Comment | Files

[19 May 12:15] Luke Quinane

Description:
When trying to fill a dataset from a stored procedure an exception is thrown if the select returns no rows.

How to repeat:
See attached file.

Suggested fix:
The exception can be avoided by wrapping the select statement in an if-block that checks for results.

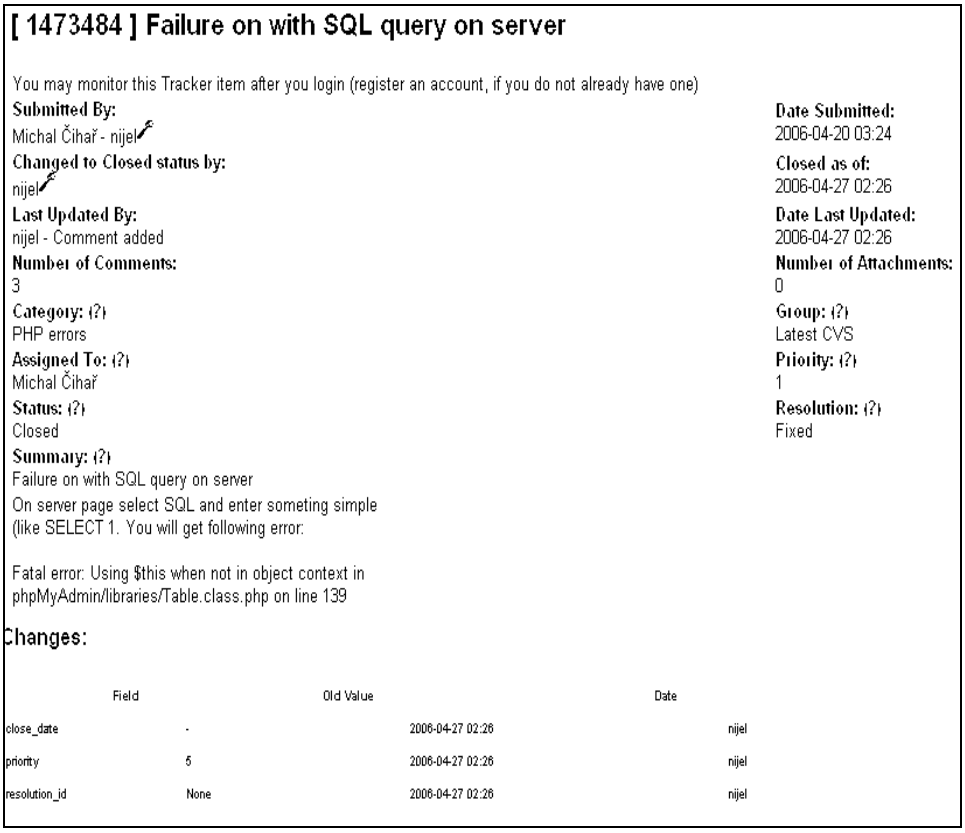
[19 May 15:09] Tonci Grgin

Luke, thanks for your complete bug report. I was able to verify it as reported on MySQL server 5.0.22bk and WinXP SP2.

[22 May 23:22] Reggie Burnett

Fixed in 1.0.8

Figure 4.7. Defect report in Eventum. [Eve06]



[1473484] Failure on with SQL query on server

You may monitor this Tracker item after you login (register an account, if you do not already have one)

Submitted By:
Michal Čihař - nijel

Date Submitted:
2006-04-20 03:24

Changed to Closed status by:
nijel

Closed as of:
2006-04-27 02:26

Last Updated By:
nijel - Comment added

Date Last Updated:
2006-04-27 02:26

Number of Comments:
3

Number of Attachments:
0

Category: (?)
PHP errors

Group: (?)
Latest CVS

Assigned To: (?)
Michal Čihař

Priority: (?)
1

Status: (?)
Closed

Resolution: (?)
Fixed

Summary: (?)
Failure on with SQL query on server

On server page select SQL and enter someting simple
(like SELECT 1. You will get following error:

Fatal error: Using $this when not in object context in
phpMyAdmin/libraries/Table.class.php on line 139

Changes:

Field	Old Value	Date	
close_date	-	2006-04-27 02:26	nijel
priority	5	2006-04-27 02:26	nijel
resolution_id	None	2006-04-27 02:26	nijel

Figure 4.8. Defect report in Tracker. [PHP06]

We can see from the figures that Eventum and Tracker have all the major attributes that were presented earlier in Table 4.4. However, Eventum does not have types for the defects, although it is possible to use severity to express enhancement requests. A major improvement in Eventum is the integration to VMS, more specifically to CVS. Tracker, in contrast, does have categories for defects, enhancements, support requests and patches as default.

Although these DMSs are quite similar, they store slightly different information about defects. The features and attributes of DMS are shown in Tables 4.6 and 4.7. The features are separated to correspond to Table 4.1. The attributes were divided into groups that correspond to those in Table 4.4. However, this list is not complete, because DMSs store additional information that is not presented in the default form of the defect reports.

Table 4.6. Feature comparison of DMSs.

	Bugzilla	Issue Tracker	KDE Bug Tracker	Eventum	Tracker
Defect reporting					
Enhancements	X	X	X	X	X
Support requests		X			X
Defect status management	X	X	X	X	X[1]
Assignment of defects					
For whom	X	X	X	X	X
Status	X		X		
Defect resolution management	X	X	X	X	X

1) Only the statuses open, pending, closed and deleted are available.

Table 4.7. Attribute comparison of DMSs.

	Bugzilla	Issue Tracker	KDE Bug Tracker	Eventum	Tracker
Id	X	X	X	X	X
Environment	X	X	X	X	X
Status	X	X	X	X	X
Resolution	X	X	X		X
Assigned to	X	X	X	X	X
Severity	X		X	X	
Reporter	X	X	X	X	X
Summary	X	X	X	X	X
Type/Classification		X			X
Change set				X	
Activity log	X	X	X		X

As we can see from the tables, the features and attributes are quite similar in these DMSs. Two major differences are the lack of an activity log in Eventum, and support for different types of defect reports in Tracker and Issue Tracker. The lack of an activity log makes it impossible to model and analyse the workflows of the maintenance processes in the OSSPs that use Eventum.

5 VERSION MANAGEMENT SYSTEMS

The software developers have worked with the source code for long time. In the software development, the source code is modified daily but sometimes changes are not good, so developers must be able to restore older versions of the source code. Moreover, nowadays when many developers around the globe work concurrently, modifications have to be merged in a controllable way. A version management system (VMS) is used to fulfil these requirements. In some contexts, a VMS is also called a version control system or source code management system.

VMSs are nowadays considered to be a part of software configuration management (SCM) and configuration management (CM). Configuration management is defined as [IEE04, p. 7-1]:

> "A discipline applying technical and administrative direction and surveillance to: identify and document the functional and physical characteristics of a configuration item, control changes to those characteristics, record and report change processing and implementation status, and verify compliance with specified requirements."

However, the difference between CM and SCM can be unclear, but they are often used concurrently. When software is developed parallel to hardware items, SCM takes place with software activities and CM takes place with hardware activities [IEE04, p. 7-2].

Asklund [Ask02] considers CM from two different perspectives, those of management and of the developer. From the management perspective, the purpose of CM is to direct and control the maintenance by controlling and identifying components and changes in them. From the developer perspective, the purpose of CM is to maintain current components and store their history. There are several aspects of CM to be considered, such as [Ask02]:

- Version control (version management)
- Configurations
- Concurrency control
- Build management
- Release management
- Workspace management
- Change management

Asklund argues that version management is a key feature in CM; it is the core functionality in several CM tools. From our viewpoint, it is necessary to look more closely at aspects of version management and change management, and leave other aspects out of further analysis. [Ask02]

As shown in Section 3, the most common input for the change process should be a defect report or a modification request. Because the change processes uses VMS, Section 5.1 describes relationship between VMS and DMS. Then Section 5.2 presents the usage of

VMS as a data source in the evaluation of the maintenance process. Section 5.3 compares some popular VMSs.

5.1 Relationship between VMS and DMS

The purpose of VMS is to create an environment for controlling software configuration in the concurrent development, testing and maintenance processes. VMSs store each change, so every revision is retrievable.

Table 5.1. Features of VMS.

Function	Usage
Source code checkout	Source code check out allows the developer to lock files which he is modifying, so other developers cannot modify them.
Source code revisioning	Revisioning allows developers to create source code for new revision.
Source code commit (submission)	Submission merges changes to the software's source code.
Logging	Logging allows the tracing of changes made to files.
History	History lists modules or files which have been changed lately.
Support for projects	One system can manage multiple projects without mixing up their configuration items.
Multi-user	Support for multiple simultaneous users.
Branching	Support for branches of the source code.
Atomic commits	Changes to multiple files submitted simultaneously by one author are made as a transaction.
Change sets	A difference between any two versions of software is available as one set of changes.

The data are stored in a VMS as configuration items, which are usually files. The most common configuration item is a source code file in OSSPs. When configuration items are modified, a new revision of the configuration item is created. Configuration items can have multiple modifications between versions, which are published revisions. However, one version of the configuration item can be incompatible with some target systems, so several parallel revisions and versions can be kept in branches. The branches are parallel development lines of the configuration items and software. The source code of the configuration item is duplicated, so different branches can be developed or maintained separately. [APC98]

The software has usually at least two parallel source code branches in OSSPs. The first branch is a stable version, which is maintained and only small enhancements are implemented. The second branch is the development version, which is under evolution and all new enhancements are implemented there.

However, if all revisions and versions were stored as complete configuration items, the amount of storage space required would be huge. Therefore, the changes in the configuration items are stored as deltas. A delta is the difference between two configuration items. Usually, the size of the delta is much smaller than the size of a new version of a changed configuration item, because the new version is derived from its predecessor. [CMB06]

In the Open Source Software maintenance process framework, it is assumed that a defect report causes a change of the source code. The changes can be classified by this characteristic. This classification categorizes changes of the source code into two groups: defect-initiated and non-defect-initiated changes. A defect-initiated change of the source code has followed the described maintenance process, but a non-defect-initiated change of the source code has not followed the maintenance process described in the framework or is documented incorrectly. Figure 5.1 shows how defects are coupled to the changes of source code.

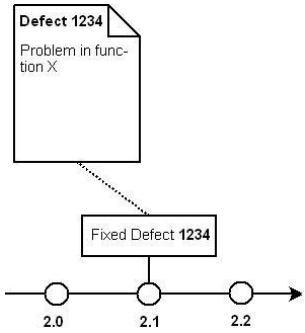


Figure 5.1. Link between changes and defect reports. [SZZ05]

The box represents the software modification and the document is a defect report. When the change includes the number of a defect and a sign that defect was fixed, the change can be assumed to be initiated by the defect report.

However, the approach has some problems. First, the description of the changes can be quite ambiguous in OSSPs. In addition, the quality of descriptions can be quite low because it depends on multiple factors such as the developer and project. In spite of the quality variations, most of the software changes can be classified as defect-initiated and non-defect-initiated changes for further analysis. The details of the classification method are presented in more detail in Section 5.2 and Paper V. Sliwerski et al. [SZZ05] have described a similar approach.

5.2 A VMS as a data source

A CVS includes all the essential attributes and functions presented in Table 5.1. From our viewpoint, the logging and history functions are the most interesting ones. The logging and history should store at least the attributes shown in Table 5.2 [Fog90, CVS05].

In addition to these attributes, the initial version of each configuration item has to be stored. Later versions can be stored as deltas.

Table 5.2. Attributes of the changes of the source code in a VMS.

Attribute	Explanation
Date and time	When was the modification submitted?
Submitter	Who submitted the modification?
Filename(s)	What or which files were modified?
Revision(s)	Are there new revisions of the modified files?
Branches	What is the revision of the new branch? (if branched)
Number of added, deleted or modified lines	How many source code lines were added, modified or deleted?
Comment	Description of the change
Modified lines	What lines were added, modified or deleted, and how?

With the attributes in the table, it is possible to model evolutions of the source code or software and analyse several aspects of the changes and project such as authors, number of changes and number of changed files per submission.

Examples of CVS log entries are presented in Example 5.1. The logs are presented here to provide a basis for the description of the approach given in Section 5.1. The first box in the example presents general information about the configuration item such as the file name (`RCS file`) and number of revisions (`total revisions`). The other boxes are log entries. They present the revision, date, author, status and amount of added and deleted lines (`lines`). The italic text is a comment of the change, which should express associated defect reports.

Example 5.1. Output of the CVS log.

```
RCS file: /trunc/src/main.c
Working file: src/main.c
head: 1.10
branch:
symbolic names:
        RELEASE_1_0: 1.9
        RELEASE_0_9: 1.8
keyword substitution: kv
total revisions: 15;    selected revisions: 15
description: ----------------------------
revision 1.10
date: 2006/04/18 14:18:14;  author: MrX%foo.bar;  status: Exp;
lines: +10 -10
Changed license version (1)
----------------------------
revision 1.9
date: 2006/01/15 06:14:13;  author: MrX%foo.bar;  status: Exp;
lines: +2 -5
Fixed Bug #24321. (2) Parameters should be read correctly.
----------------------------
revision 1.8
date: 2005/10/09 01:54:06;  author: MrX%foo.bar;  status: Exp;
lines: +8 -0
Added function foo(). (3)
```

If we analyse the comments of these changes more carefully, we can see that only one of them is related to the defects reports. The comment of the change (2) expresses clearly that it was related to the defect report whose id is 24321, and it actually fixed it. Other changes such as (1) did not relate to defect reports because (1) changes only the version of the license. However, change (3) may have a relation to the defect reports because it added a new function `foo()`, which probably brings new functionality to the software. Although the comment of this change does not state that, it would be defect related. The submitter of the change might have forgotten to express this relation..

Log entities do not include the changed lines of the source code. However, every change can be examined in details in the CVS, which uses `diff`(a file comparison utility that lists the differences between two files) for recognizing changes between files. It was originally developed in the 1970s. Nowadays, there are multiple implementations and their outputs are compatible. Example 5.2 presents details of the changes between versions 1.9 and 1.8 of the file. In the example, the first box presents the name, revisions and dates of the changed file. The second box presents the source code before any changes, and the third box presents the source code after the changes. Lines marked with "-" were deleted between revisions, and lines marked with "+" were added.

Example 5.2. Output of the CVS Diff.

```
RCS file: /trunc/src/main.c
retrieving revision 1.15
retrieving revision 1.16
diff -c -r1.15 -r1.16
*** /trunc/src/main.c 9 Oct 05 01:54:06 1.15
--- /trunc/src/main.c 15 Jan 06 06:14:13 1.16
```

```
*** 49,50 ****
      static void main(ARGS[]) {
-         param1 = getParam("X;");
```

```
...  --- 49,50 ----
     static void main(ARGS[]) {
+          param1 = getParam("X");
...
```

As we can see from the example, only one character was removed from the source code. However, since the CVS and other systems handle configuration items in rows, a whole line has to be removed and added again. Example 5.3 is the log in SVN [Tig06b].

Example 5.3. Output of the SVN log.

```
r8 | MrX | 2005-07-14 08:15:29 -0500 | 1 line
Changed paths:
M /trunk/src/main.c
Fixed Bug #24321. Parameters should be read correctly.
------------------------------------------------------------
r7 | MrX | 2005-07-03 08:15:29 -0500 | 1 line
Changed paths:
M /trunk/src/main.c
M /trunk/src/main.h
A /trunk/src/doc/README
Updated License version and added README file.
```

In SVN, a change can influence multiple files, as the example shows. The attributes are a revision (`r8`), author (`MrX`), date, changed file (`M /trunk/src/main.c`) and comments. The attribute *revision* expresses the identification number of the revision, which is sequential. The attributes *author* and *date* express who made the changes and when. The attribute *changes files* expresses which files were modified (`M`), added (`A`) or deleted (`D`). The attribute *comment* indicates the purpose of the changes and possible connection to the defect report.

Example 5.3 showed that the log entries do not include changed lines of the source code. However, SVN is able to provide *diff*-stylish differences between two revisions, as shown in Example 5.4.

Example 5.4. Output of the SVN diff.

```
Index: main.c

--- main.c (revision 1.8)
+++ main.c (revision 1.9)
@@ -1,50 +1,50 @@

    static void main(ARGS[]) {
-         param1 = getParam("X;");
+         param1 = getParam("X");
...
```

In the example, the box includes the name (`main.c`) and revisions of the file (`revision 1.8` and `1.9`), and these are compared. In addition, it presents the file before and after changes. Lines that were deleted are marked with "-" and lines that were added are marked with "+".

5.3 Example systems

VMSs were also used in early software development. After manual management, the first management systems were designed for personal use with one or a few files. These first generation VMSs, such as the Revision Control System (RCS), became too limited when network connections become common. The second generation VMSs, such as the Concurrent Versions System (CVS), then replaced earlier systems. However, these systems also had limitations and new third and fourth generation systems, such as the Subversion (SVN) and the Git, were created.

The Revision Control System (RCS) was one of the pioneers of VMSs. It was developed in the 1980s at Purdue University, and automated the storing, retrieval and logging of revisions. However, it was replaced later because it was unable to manage whole projects and multiple simultaneous users. [RCS06]

The weaknesses of RCS led to the development of CVS, which is now the most widely used and best known VMS. Its most important improvements were support for projects

and multiple users. CVS was developed by Brian Berlinger in 1989. It is Open Source licensed, and spread to multiple platforms such as Linux, Windows and Unix [Fog90].

Nowadays Subversion (SVN) is replacing CVS as a VMS. The development of SVN was started in early 2000, when the limitations of CVS were found to be too disturbing. SVN was created from scratch to avoid the limitations of CVS but includes advantages. CVS was designed for concurrent distributed software development but the source code could be corrupted if two developers submitted modifications at exactly the same time. The modifications to the files were received from the first developer and therefore further modifications to the same file could not be received if these developers were working with the earlier version of the file.

SVN resolves this problem with atomic commits. An *atomic commit* is a set of changes that are considered as one transaction. If there is an error during an atomic commit or some parts of a transaction fails, the whole transaction is cancelled. Therefore, either all changes are made or none of them is made.

Since the source code is modified daily, there is usually more than one commit between two releasable versions and therefore commits have to be grouped into change sets. A change set is a list of the changes between two different versions of the software. For example, CVS does not have software versions and manages individual files so cannot support changesets. [CMB06]

However, not all users were satisfied with SVN because it had client-server architecture instead of distributed architecture. This dissatisfaction led Linus Torwalds and Junio Hamano to creat Git [Git06], which is a modern VMS that has a decentralized architecture. It was initially created for the development of Linux kernel. In addition to decentralized architecture, it has many other improvements such as support for non-linear development and cryptographic authentication. It was made more time efficient on large-scale projects such as Linux kernel, and it stored whole files instead of the changes. However, support for storing changes was added later because the requirement for storage space increased too much in some projects. [Git06]

Although distributed systems provide benefits, they also have drawbacks. With a distributed system, the maintenance process is more difficult to manage, the forks of the software can be more common, and the changes are not well identifiable.

Next, we will provide a brief comparison of these four systems through features that were mentioned earlier. The comparison is presented in Table 5.3.

Table 5.3. Comparison of RCS, CVS, SVN and Git.

	RCS	CVS	SVN	Git
Product information				
Licence	Multiple	GPL	Apache/BSD stylish	GPL
Released	1980	1986	2002	2005
Repository architecture	Client-server	Client-server	Client-server	Distributed
Features				
Source code checkout	X	X	X	X
Source code revisioning	X	X	X	X
Source code commit	X	X	X	X
Logging	X	X	X	X
History	X	X	X	X
Support for projects		X	X	X
Multi-user		X[2]	X	X
Branching		X	X	X
Atomic commits			X	X
Change sets			X[1]	X

1) Only between two sequential versions
2) Because there are no atomic commits, some problems may occur

As we can see from the table, RCS and CVS are much older than the other two options. They have been used as examples when newer systems have been developed. CVS does not have atomic commits or change sets like SVN and Git does, but still CVS is used much more widely than the other systems. The lack of support for atomic commits creates a problem when analysing commits from CVS. If a developer modifies several files at once, these modifications have to be considered as one atomic change. An author, comment of the change and time and date of the change are used to group the modifications. With this method, the logs of CVS can be compared with logs of other systems.

In addition to these four VMSs, there are multiple other VMSs such as:

- Aegis [Aeg06]
- Bazaar-NG [Baz06]
- Darch (David's Advanced Revision Control System) [Dar06]
- GNU Arch [Arc06]
- Mercurial [Mer06]
- Monotone [Mon06]
- OpenCM [OCM06]
- svk [SVK06]
- Vesta [Ves06]

6 OSS MAINTENANCE PROCESS EVALUATION

There are two concurrent management processes in the maintenance process, DM and CM. These processes are connected to each other because they produce and consume deliverables of each other. The CM process is started when a defect report is assigned and it should finish when the feedback is provided to the DM process. Therefore, we analyse changes and their connection to the DM process.

The control is based on measurements, and improvements are based on the results of the measurements. Fenton et al. have defined measurements in the following way [FeP97]:

> "A process by which numbers or symbols are assigned to attributes of entities of the real world in such a way as to describe them according to clearly defined rules."

The *entity* is an object, such as a person or room, or event, such as a journey or the testing phase, and the *attribute* is a feature or property of the entity. The measurements are connected together with relations that can be empirical or numeric.

Fenton et al. [FeP97] propose three entity types that could be measured in software engineering: the process, product and resources. The process is a collection of related activities; products are results of the process such as artifacts, deliverables or documents. Entities that are required by any of the process activities are resources. The most common attributes for the process are the duration, effort and number of incidents. The most common attributes for the product are the size and quality aspects. The most common attributes for the resources are the cost, magnitude and quality.

Although we have tools for presenting measurements and for calculating statistics and figures, measuring is meaningless if we do not know what to measure. One widely recognized approach is the goal-question-metric (GQM) approach [BCR94]. The goal describes the purpose of the measurements. The questions are derived from the goals and their answers provide information that is required to decide whether the goal is archived. Metrics are results of the measurements that provide information for answering questions. A metric is usually defined as a controlled measurement system with base values and interpretation for evaluating attributes. Metrics have base values (measurements) that are used to define their domains and indicators. For example, one metric, which describes the activity of the maintenance process, could be the number of new defects that are reported in a month. The metric is measured by counting each of the defects that are reported and the results are grouped by months, for example. The metric interprets the value of the measurement with a model that is associated to the metric, which in this case is higher value is related to the more activite process. As a result, this metric makes it possible to measure the activity (which was the out attribute) of the maintenance process (which was our entity).

The third research objective was to create an approach and framework for the evaluation of the OSS maintenance process. Therefore, we present attributes and metrics for the analysis of OSS maintenance processes. Paper VI presents original research and a framework, but this section presents an elaborated description of the *evaluation framework for Open Source Software maintenance*.

Fenton et al. [FeP97] propose that the process, product and resources are measurable entities. In this thesis, we consider only process entities. We divide our attributes and metrics into three sets that measure process from the viewpoints of process activity, workflow and management. Attributes that measure resources were left out because they are not practically measurable in volunteer groups such as OSSPs.

The first set of attributes and metrics describes the activity of the process. The GQM approach for process activity is:

- Goal: Analyse the activity of the process
- Questions: How often are the activities of the process performed?
- Metrics: Metrics and attributes are described in Section 6.1.

The second set of attributes and metrics presents the maintenance process by modelling the workflows of the defects. The GQM approach for these metrics is:

- Goal: Analyse maintenance process activities
- Questions: What is the life cycle of the defects?
- Metrics: Metrics and attributes are described in Section 6.2.

The third set of attributes and metrics describes the role of the DMS in the maintenance process. The GQM approach of this viewpoint is:

- Goal: Analyse the connection between DM and VM
- Questions: How many changes are induced by defect reports?
- Metrics: Metrics and attributes are described in Section 6.3.

6.1 Process Activity

The process activity aspect describes the overall activity of the process. The first set of metrics, which illustrates the activity of the maintenance process in evaluated OSSP, is presented in Table 6.1. The number of modification requests, which are the same as defect reports in this case, was used in [GHJ04] and the number of defects has been used in a proprietary environment [CuS97] and in several DMSs such as in [Tra06]. The number of changes was used in [Mas05] and [RKG04].

Table 6.1. Metrics for the activity of the process.

Metric	Abb.	Type	Domain
Opened defects	OD	Absolute	≥ 0
Resolved defects	RD	Absolute	≥ 0
Fixed defects	FD	Absolute	≥ 0
Changes of the source code	CS	Absolute	≥ 0

While these four metrics illustrates the scale and activity of the maintenance process, they also describe the sampled data. A large difference between the number of fixed defects and the changes of source code or a large difference between the number of opened defects and resolved defects expresses a potential problem in the defect management process. A sample metrics from the Apache project is presented in Table 6.2.

Table 6.2. Metrics for the activity of the process in the Apache case study between September 2003 and August 2005

Metric	Apache
Opened defects	1266
Resolved defects	943
Fixed defects	288
Changes of the source code	2877

As we can see from the table, there were on average fewer than two opened defects a day, and fewer than four changes in the source code per day. The majority of the defects were resolved but only less than 20 per cent of them were actually fixed. But more interestingly, the number of changes is over two times higher than the number of defects and ten times higher than the number of fixed defects. These results show that most of the defects do not cause fixes and most of the changes do not seem to be originated from defects.

The second set of the process activity metrics includes the trends of process activity. These metrics are presented in Table 6.3.

Table 6.3. Metrics for the trends of process activity.

Metric	Abb.	Type	Domain
Monthly opened defects	MOD	Absolute	≥ 0
Monthly resolved defect	MRD	Absolute	≥ 0
Monthly fixed defects	MFD	Absolute	≥ 0
Monthly changes of the source code	MCS	Absolute	≥ 0

They describe the evolution of the activity of the project. An example of the results is presented in Table 6.4. Even the results presented in the table 6.2 gave the impression that most of the defects were resolved, the table 6.4 shows that the defect management activities are seasonal in the Apache project. MOD is quite constant, between 35 and 60 per month. MCS is also quite constant, between 90 and 200 per month. However, MRD and MFD are zero before November 2004, when a huge number of defects were resolved. After that, MRD and MFD stabilized to a level that is slightly lower than MOD, except in March 2005 when a huge number of defects were also resolved. As a conclusion, these results show that defect reports were not processed actively before November 2004.

Table 6.4. Trend metrics of the Apache case study.

Month	MOD	MRD	MFD	MCS
04-09	65	0	0	197
04-10	47	0	0	100
04-11	40	1494	563	224
04-12	42	25	7	114
05-01	35	36	19	123
05-02	53	43	13	104
05-03	48	394	117	74
05-04	56	35	10	96
05-05	65	44	16	87
05-06	43	68	26	97
05-07	34	10	3	95
05-08	43	83	19	147

It is possible to analyse workflow of the process through events (presented in Section 4.2). Fenton et al. [FeP97] argue that the duration, effort and incidents are common metrics for the process. We model the workflow of the process and compare results with the expected life cycle (presented in Section 3.2). In addition to process modelling, we include duration in the analysis. We distinguish duration of defects that caused changes and those that did not cause changes. Even though Fenton et al. proposed that effort should be used in metrics for the process, we exclude those attributes because it is not possible to measure effort accurately and reliably from volunteers.

6.2 Process Workflow

The process workflow aspect characterizes process by modelling workflows in defect management. These metrics describe the common defect life cycles in the analyzed project. A defect life cycle describes the workflow of defect management. Ripoche et al. [RiG03b] presented defect life cycles as metrics for measuring OSSPs. In addition to the life cycle, the duration of the life cycle is also informative. The duration of defect resolving was proposed in [CoW06], [MFH02], [KiW06] and [KLH06] as a metric. However, [CoW06] and [KiW06] presented only resolving times for defects that were fixed, while [MFH02] and [KLH06] also presented resolution time for defects that did not cause changes to the source code. These metrics are presented in Table 6.5.

Table 6.5. Metrics for process workflow.

Metric	Abb.	Type	Domain
Statuses of life cycle	ST	Nominal	Any string
Number of defects in the life cycle	LC	Absolute	≥ 0
LC / All	LC%	Ratio	0−100%
Duration of all defects	DurAl	Ratio	≥ 0 days
Duration of non-fixed defects	DurNF	Ratio	≥ 0 days
Duration of fixed defect	DurFD	Ratio	≥ 0 days

The number of defects in the life cycle (`LC`) shows how many defects that life cycle had. `LC%` presents the proportion of the defects that went through the described life cycle (ST).

The results are presented in the table ordered according to the popularity of the life cycle. These metrics provide a possibility to trace the maintenance process. If the defects do not follow the described life cycle, it can be assumed that the process is poorly managed. An example of the result is given in Table 6.6. `Med` stands for the median, and `Avg` stands for the average.

Table 6.6. Metrics for process workflow in the Apache case study.

LC	**LC**		**LC%**
New, Assigned, Resolved, Closed	36		4 %
New, Resolved	247		26 %
New, Resolved, Closed	511		54 %
	DurAl	**DurFD**	**DurNF**
Avg	98	122	82
Med	36	65	24

The results in the table give the impression that defects are rarely assigned; over 80% of the defects are resolved without assignment. The approach allows the user to study the resolutions of the defect with the selected life cycle. In the case of the Apache project, it turns out that 142 of those 511 defects had the life cycle `New,Resolved, Closed,` and the resolution was fixed. It shows that even the defects that led to changes were not assigned. Overall resolution time for the defects were relatively long, even the not-chance inducing defects were resolved a bit quicker.

However, the actual duration of the resolution does not necessarily correspond to the time taken for the users to get a modified version of the software. The modified version is available from VMS but most users wait for the release of installable software, which can be much later. Usually, the time between the modification and release of the installable software depends on the severity of the defect.

6.3 VM process workflow and management

The VM process workflow and management aspect characterises the role of DMS in the process. These metrics allows the evaluating of the relationship between the defect management process and software modification. The number of changes of the source code has been used as a metric in several studies such as [MFH02]. Metrics are presented in Table 6.7.

Table 6.7. Metrics for VM process workflow and management

Metric	Abb.	Type	Domain
Defect-initiated changes	DIC	Absolute	≥ 0
Proportion of fixed defects	FD%	Ratio	0−100 %
Proportion of defect-initiated changes	DIC%	Ratio	0−100 %
Monthly defect-initiated changes	MDC	Absolute	≥ 0
Proportion of monthly defect-initiated changes	MDC%	Ratio	0−100 %
Changes per author	CPA	Absolute	≥ 0
Defect-initiated changes per author	ADC	Absolute	≥ 0
Proportion of defect-initiated changes per author	ADC%	Ratio	0−100 %

The low percentages of `DIC%`, `MDC%` and `ADC%` indicates that the maintenance process is not managed by defect management. It may also indicate that the relationship between defect management and changes of the source code is unreliable. As an example, Tables 6.8 and 6.9 present metrics that were measured from the Apache project.

Table 6.8. Metrics for the VM process workflow and management in the Apache case study.

Metric	Apache
Proportion of fixed defects	31 %
Changes of the source code[1]	2877
Defect-initiated changes	276
Proportion of defect initiated changes	10 %

1) see Table 6.2

Table 6.9. Metrics for the VM process workflow and management in the Apache case study

Month	MCS [1]	MDC	MDC%
04-09	197	26	13 %
04-10	100	14	14 %
04-11	224	10	4 %
04-12	114	7	6 %
05-01	123	7	6 %
05-02	104	10	10 %
05-03	74	1	1 %
05-04	96	5	5 %
05-05	87	7	8 %
05-06	97	12	12 %
05-07	95	2	2 %
05-08	147	9	6 %

1) see Table 6.4

As we can see from the table 6.9, MDC% is very low during the evaluation period. Since the proportion is so low, most of the changes are not related to defects. The table shows that maintenance is not managed with VMS in the Apache project. The low MDC% could also be due to non-mature software or poorly commented changes; in any case, it is still a sign of an unmanaged maintenance process.

Because of the low proportion, we should also look at statistics of each author separately. As an example, Table 6.10 presents metrics that were measured from the Apache project.

Table 6.10. Metrics for the VM process workflow and management in the Apache case study.

Developer	CPA	ADC	ADC%
Author 1	197	26	13 %
Author 2	100	14	14 %
Author 3	224	10	4 %
Author 4	114	7	6 %
Author 5	123	7	6 %
Author 6	104	10	10 %
Author 7	74	1	1 %
Author 8	96	5	5 %
Author 9	87	7	8 %
Author 10	97	12	12 %
Author 11	95	2	2 %
Author 12	147	9	6 %
...			

As we can see from the table, these developers act quite similarly, since ADC% varies from one to 14 percent. It may be that some authors are not motivated to express linkages to defect reports, or may not be familiar with procedures.

Usually, software has many developers, so we also analyse differences between developers and developer groups. For example, in Paper VII, we grouped authors according to activity into *developers* and *users*. Developers had made more than 20 changes to the source code in the last two years, and users had made fewer than 20 changes. In the Apache project, developers made fewer defect-initiated changes than users, and the difference was statistically significant ($p<0.05$ with the independent T-test). The difference might be explained by the different roles. Furthermore authors, might have different types of roles, or perhaps new developers follows guidelines more precisely than do more experienced ones.

7 REMOTE ANALYSIS SYSTEM FOR OPEN SOURCE SOFTWARE

Since DMSs and VMSs contain very large amounts of data, automated systems would make analysis possible in reasonable time. There existed systems such as SoftChange, GlueTheos and CVSAnalY that retrieved and analysed the data from these systems, but none of them integrated the data from both systems. Antoniol et al. have proposed a model for the integration [ADH04]. SoftChange was created by German et al. [Ger04, GHJ04] when they studied CVS repositories. It visualizes the evolution of the software. Robles et al. created GlueTheos [RGG04] which retrieved data from CVS and delivery packages. Its purpose was to analyse the source code and its complexity. In addition to GlueTheos, Robles et al. also created CVSAnalY [RKG04], which retrieved data from CVS and provided historical and real-time data about source code and its contributions. German et al. [Ger04] claim that SoftChange retrieves version management and defect management data, but only the version management data was used in the analysis.

Therefore, we have created a prototype of an automatic system that retrieves and analyses data from DMSs and VMSs. Our system is primarily meant for people who are neither involved in OSSPs or potential users of the software. Since the users are outside the OSSP, they do not have direct access to the defect management or change management databases. Our system retrieves data by using methods that are available to anyone, not only project developers. The purpose of the system is to provide an overview of the maintenance process and key metrics, so users could evaluate whether to acquire and use the software or not. The main purpose of the metrics is to provide information that could be used for deciding whether the users could rely on the maintenance of the software. Since our software is meant for users, we also provide it for academic and scientific purposes, so it could be used to understand OSS maintenance processes. In this section, we will briefly present the architecture (7.1), database (7.2) and user-interface (7.3) of the Remote analysis System for Open Source Software (RaSOSS).

7.1 Architecture

Since OSSPs use different systems for defect management and version management, we had to provide support for several DMSs and VMSs. Therefore, the basic requirements for the system are extensibility and independence from these management systems. To achieve these requirements the architecture makes a distinction between data collection and analysis (Figure 7.1). The distinction allows the adding of new data sources and analysis modules to the system. RaSOSS has been programmed with PERL and it uses GTK+ for graphical interface.

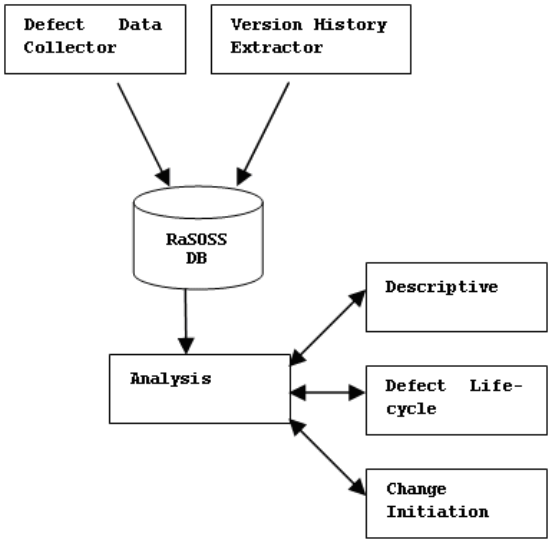


Figure 7.1. Design architecture of RaSOSS.

The figure shows that the architecture of RaSOSS is a mixture of typical pipe-filter and blackboard architectures. The three main modules in the system are the defect data collector, version history extractor, and analysis module. The defect data collector retrieves defect data from DMS. The version history extractor retrieves the source code of the software and the change history of source code files from VMS. After the data retrieval, both modules export data to the database where the analysis module can classify and analyze the defects and changes.

7.2 Database

The database schema records information in an original system independent form. The database is organized as a collection of entities and relationships stored in a relational database management system. The relational structure of the RaSOSS database structure is shown in Figure 7.2 and the attributes are described in Tables 7.1 and 7.2. The database structure has two parts because DMS and VMS are not integrated and there is only a weak connection between the systems, as shown in Section 5.1. The first part of the database is based on the entity `Defect` and the second part is based on the entity `Revision`.

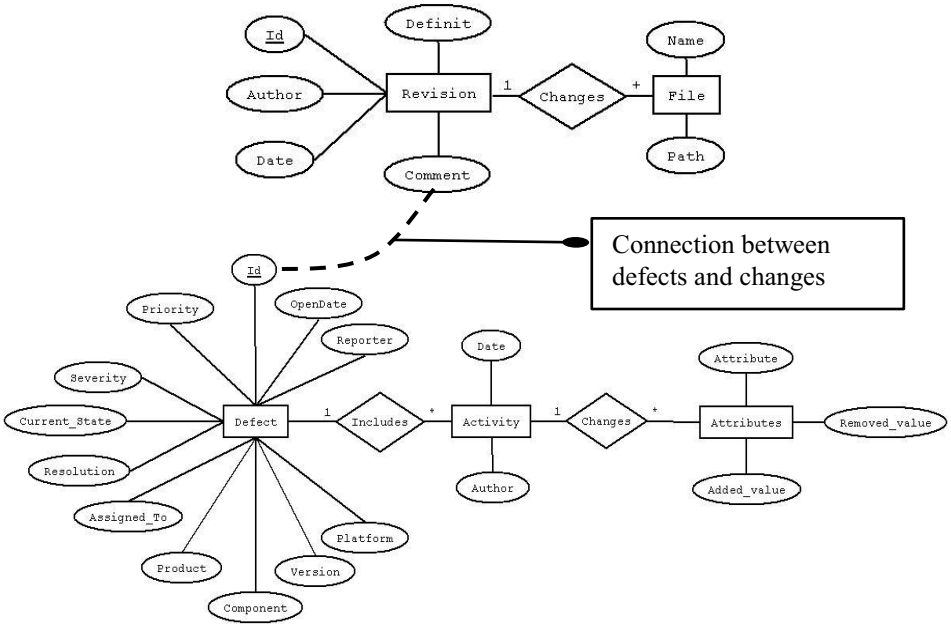


Figure 7.2. Database structure for RaSOSS.

The entity `Defect` describes attributes that are directly related to the defect report. The attribute `Reporter` expresses the name, nickname or email address of the person that reported the defect. The attribute `Opendate` expresses the date when the defect was opened. Current status describes the defect's status such as new or resolved. `Resolution` expresses the outcome of the defect. `Component`, `Product`, `Version` and `Platform` describe the environment where defects appear. Entities `Activity` and `Attributes` express when, by whom and what attributes were changed in the defect report.

A revision of the software is a snapshot in the evolution. Every revision has an attribute to describe the `Author`, `Date` and `Comment` of the revision. The comment describes the authors' comment about the changes between revisions. The attribute `Definit` describes whether the changes between revisions were initiated by defect reports. The entity `file` and its attributes describe what source code files were changed from the previous revision.

Table 7.1. Attributes of the defect in the RaSOSS database.

Entity/Attribute	**Type**	**Data type**	**Usage**
Defect			
Id	Ordinal	Integer	Identification number of the defect report
OpenDate	Interval	Date	Date when the defect report was opened
Priority	Ordinal	Integer	Priority expresses urgency of the defect
Severity	Ordinal	String	How severe is the defect? For example Major, Critical, Minor etc.
Reporter	Nominal	String	Who reported the defect initially?
Current_status	Nominal	String	What is the current status? see Table 4.5
Assigned_to	Nominal	String	Who is working/ worked on the defect?
Resolution	Nominal	String	How was the defect resolved? see Table 4.2
Product	Nominal	String	Indicates in which software, component, version and system platform the defect occurs.
Component	Nominal	String	
Version	Ordinal	String	
Platform	Nominal	String	
Activity			
Date	Interval	Date	When were the attributes of the defect report changed?
Author		String	Who made the changes?
Attributes			
Attribute	Nominal	String	Which attribute was changed?
Removed_value	Nominal	String	What value was removed?
Added_value	Nominal	String	What value was added?

Table 7.2. Attributes of the revision in the RaSOSS database.

Entity/Attribute	**Type**	**Data type**	**Usage**
Revision			
Id	Ordinal	Integer	Identification number of the revision of source code
Date	Interval	Date	When was the revision created?
Author	Nominal	String	Who created the revision?
Comment	Nominal	String	How did the creator describe changes from the previous revision?
Definit	Nominal	Boolean	Were the changes of the revision initiated by the defect report?
Files			
Name	Nominal	String	Filename
Path	Nominal	String	Path

7.3 User Interface

RaSOSS is a set of PERL scripts that are executed mainly sequentially and partly concurrently. Manual execution of the scripts is error prone and time consuming, so we built a user interface for our system. The user interface of RaSOSS was designed to be as simple as possible and highly automated. The only required inputs from the user are the addresses of DMS and VMS, product names and database used.

The user interface has been divided into three screens. The first one is a start-up screen, the second and third screens are main screens. Figure 7.3 shows the start-up screen.

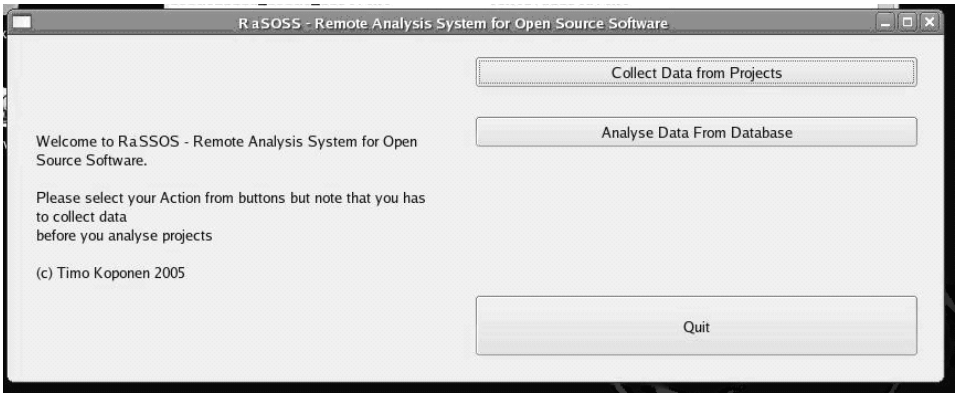


Figure 7.3. User interface of Start-up.

The button `Collect Data from Projects` starts the data collector module. The button `Analyze Data From Database` starts the data analysis module. Figure 7.4 shows the user interface of the data collector module.

Before any other action, the user has to fill in the name of the database which will be used to the field `Database Name`. If the database is empty, the user can create tables with the button `Clear database`. The user can also clear the database with the same button, which will drop tables from the database if there are any, and create new tables.

After the database initialization, the user can continue to the data retrieval, but before the data retrieval from VMS, the user has to fill in the URL of VMS in the field `SVN/CVSROOT`. In addition, the user has to select the type of system and fill in the name of the product in the field `Product name`.

Before the data retrieval from DMS, the user has to fill in the URL and type of target DMS. In addition, the user has to know the name of the product and fill it in the field `Product name`.

Then the user can retrieve data from VMSs (button `Get CVS Data`) and DMS (button `Get Bug data`). However, one minor issue in the retrieval of the defect data is that the

execution time can be more than ten hours (with 1.5 Ghz Pentium-M laptop, with 512 Mb ram and 1Mb/s internet access) if the number of defects is over 30 000, because each defect is retrieved separately. Although the retrieval method could be executed in parallel fashion, this would increase the load of the target DMS and VMS, so it is not desirable.

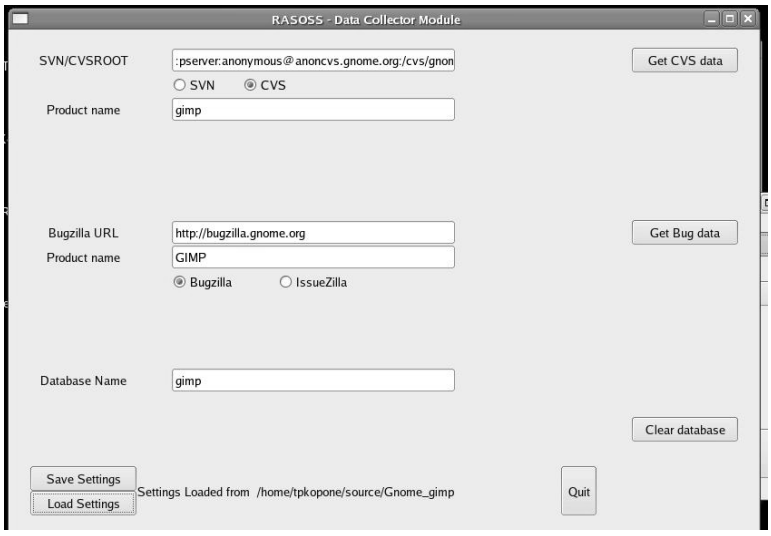


Figure 7.4. User interface of Data Collector module.

The module is also able to save and load settings (buttons `Save Settings` and `Load Settings`). The textfield on the right side of the load and save setting shows the response of the last action.

Figure 7.5 shows the user interface of the data analysis module. In the data analysis module, the user has to select the database appropriate for the analysis to be performed. In addition, the user has to select the sets of the attributes and metrics that will be calculated. The result of the analysis is presented in an HTML document.

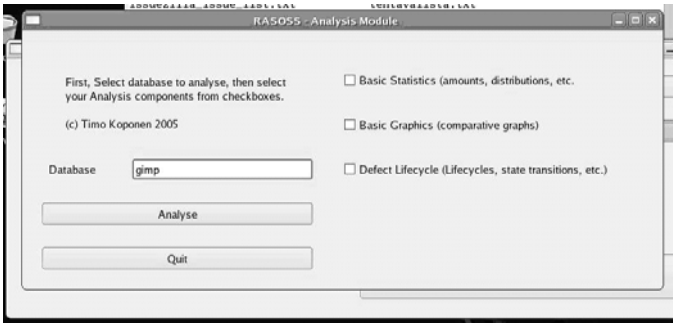


Figure 7.5. User interface of the Analysis module.

8 CASE STUDIES

During the research, we analyzed two case studies, the Apache HTTP Server and Mozilla Firefox, manually and later, after the development of RaSOSS, we reanalyzed the Apache HTTP Server [Apa06], and Mozilla Firefox [Moz06a] case studies to validate RaSOSS. In addition, we analysed Gnu Glib [Gli06], Gimp [Gim06], KDE KWord [KOf06], OpenOffice.org Calc [OOo06d], and Writer [OOo06f] as new case studies. This section presents the results of the case studies briefly: the original research and results are presented in Paper VII. We collected defect reports and the version history of the source code from each project. Our snapshots are limited to between September 2003 and September 2005.

We selected case studies to cover a wide range of software types and target groups. Mozilla Firefox, KDE Kword and Gimp are intended for end-users. Mozilla Firefox and KDE Kword are meant for almost everyone, but Gimp is meant for digital imaging amateurs and professionals. The Apache HTTP Server is meant for system operators and administrators. Gnu Glib is meant for software developers who need basic data structures and functionalities. General information about the case studies is presented in Table 8.1.

Table 8.1. General information about the case studies.

	Apache	**Firefox**	**Gimp**	**Glib**	**Kword**	**Calc**	**Writer**
Software type	Server software	Desktop software	Desktop software	System library	Office software	Office / spreadsheet	Office / word processing
Intended audience group	Admin/operator	End-user	End-user	Developer	End-user	End-user	End-user
License and its group	Apache, Licenses without copyleft	MPL, Licenses with privileges	GPL, Licenses with strong copyleft	LGPL, Licenses with restricted copyleft	GPL, Licenses with strong copyleft	LGPL, Licenses with restricted copyleft	LGPL, Licenses with restricted copyleft
DMS	Bugzilla	Bugzilla	Bugzilla	Bugzilla	KDE Bug tracker	Issue Tracker	Issue Tracker
VMS	SVN	CVS	CVS	CVS	CVS	CVS[1]	CVS[1]

1) Transition to SVN is being planned.

The Apache HTTP Server was originally created in the National Center for Supercomputing Applications by Rob McCool. In 1994, the development stalled and a variety of

improvement patches started to spread. In 1995, the development started to act like an OSSP, the website was created and the software was renamed Apache. [Apa06]

The Mozilla Firefox project started in 2002, when it branched from the original Mozilla web browser. Mozilla was based on the source code of Netscape, which was released in 1998 by the Netscape Company. The release led to the establishment of the community that started to develop Mozilla projects. Version 1.0 was released in September 2004 and it gained 10 percent of the market share globally. [Moz06a]

Gimp was originally created in the University of Berkeley by two students, Kimball and Mattis. The first public version was released in 1996, and shortly after the release users started to write websites and tutorials. New features were implemented daily. Later the name was changed from General Image Manipulation Program to Gnu Image Manipulation Program. In 1997 the original developers graduated and quit development. Thereafter, development shifted more closely to the OSS development model and DMS was initiated. [Gim06]

Gnu Glib is a cross-platform utility library. At first, it was a part of GTK, Gnu Graphics Toolkit, but it came to be used in other applications. Currently, Glib includes several functionalities, such as threads, memory allocation and type conversions, and data types such as strings and arrays. [Gli06]

KDE Kword is a free WYSIWYG-type word processor and it is a member of the KOffice project. The first version was created in 1998. In 2000, KWord was hardly maintainable and no one was working on the known problems any longer, but in the same year new maintainer started to fix and restructure the source code. After 2001, Kword was stable and the number of users increased again. [KOf06]

OpenOffice.org Calc and Writer are part of the OpenOffice.org software suite. The OpenOffice.org project was started in 2000 when Sun Microsystems released the source code. Writer is word processing software that is GUI-based and very similar to Microsoft Word. Calc is spreadsheet software that is similar to Microsoft Excel. The OpenOffice.org suite also includes applications for presentations, vector drawing and databases. Because of the project heritage, project documents describe explicitly the processes that are related to maintenance and defects. [OOo06d, OOo06f]

As we showed in Section 6, our metrics were divided into three aspects - process activity, process workflow and VM process workflow and management. Next, Section 8.1 will present the results of the first aspect, Section 8.2 the results of the second aspect and Section 8.3 results of the third aspect.

8.1 Process activity

The metrics for the process activity give an overview of users' and developers' activity in the case studies. First, we present, in Table 8.2, the number of defects, changes and defect-initiated changes.

Table 8.2. Metrics for the activity of the process in the case studies.

	Apache	Firefox	Gimp	Glib	Kword	Calc	Writer
Opened defects[1]	1266	27681	3088	786	568	9227	3432
Resolved defects[2]	943	20300	2725	674	369	6271	2459
Fixed defects[3]	288	2415	1114	390	207	1985	667
Changes of the source code[4]	2877	2884	7102	1262	606	13384	8063

1, 2, 3, 4) see Table 6.1

Mozilla Firefox had had almost 170 million downloads before April 26 [Moz06b], which explains the great number of defect reports. But Apache has been installed on over 26 million public web servers, and the number of defect reports is much smaller than for Firefox (the ratio of installations: 15 % (26/170millions), the ratio of opened defects: 4.6 % (1266/27681)) [Net06]. The number of actual installations of Apache may be much higher, because it is also used in non-public web servers. The number of users of Gimp, Glib and Kword is not available, but the number of users of Glib at least is high, because it is used in many other OSS products.

The proportion of fixed defects varied from Kword's 56 % (207/369) to Firefox's 12 % (288/943). Other defect reports were duplicate, invalid or in other ways did not cause modifications to the software.

In addition to a general view of the activity, the description is more detailed when observing the same metrics in shorter periods such as monthly. Monthly metrics are presented in Appendix I.

8.2 Process workflow

OSSPs may describe their development and maintenance activities and processes in their project documentations. However, when DMS is used, it is possible to model maintenance processes in the projects by using metrics changes of the defect reports as process events, as we described in Section 4.2. Table 8.3 presents the most common defect life cycles in the case studies.

Table 8.3. The most common life cycles in the case studies.

Project	Defect life cycle (ST)[1]	LC[2]	LC%[3]
Apache	New, Resolved , Closed	511	54 %
Firefox	Unconfirmed, Resolved	11133	56 %
Gimp	Unconfirmed, Resolved	1351	49 %
Glib	Unconfirmed, Resolved	417	62 %
Kword	Unconfirmed, Resolved	257	70 %
Calc	Unconfirmed, Resolved, Closed	1302	50 %
Writer	Unconfirmed, Resolved, Closed	2987	45 %

1-3) aee Table 6.5

The table shows that the defect life cycles were not expected in the case studies. There was no significant use of the status `assigned` or `started` in any of the cases,. These statuses should indicate that someone is implementing a modification. The status of defects was transited directly to the status `resolved`, which indicates that all modifications have been implemented already.

The defect reports have to be resolved as quickly as possible to keep users satisfied. The averages (Avg) and medians (Med) of the resolution times for the fix-inducing (DurFD) and non-fix-inducing (DurNF) defects are presented in Table 8.4.

However, it is not sensible to compare the average or median resolution times of the projects with each other because the case studies concern different types of software and project organisations. The main purpose of the times is that the difference between fix-inducing, non-fix-inducing times can be compared, and the scale of the resolution times can be notified.

Table 8.4. Resolution times of the defects.

Metric		Apache	Firefox	Gimp	Glib	KWord	Calc	Writer
All	Med (days) [1]	36	3	1	4	19	14	25
	Avg (days) [2]	98	54	36	56	137	64	72
DurNF	Med (days) [3]	24	1	0	2	33	3	7
	Avg (days) [4]	81	46	23	58	132	31	53
DurFD	Med (days) [5]	66	34	3	5	13	97	70
	Avg (days) [6]	122	91	50	49	129	138	103
1-6) see Table 6.6								

Averages of the resolving time vary from the Gimp project's 23 days to the Kword project's 132 days. However, the problem with measuring the time between opening and resolving is that a defect may be reopened several times. Another problem is that many defects are resolved the same day, while others may require several months. In addition, the average is not representative because the distribution is skewed.

Medians of resolving times show the time where half of the defects were resolved. Table 8.4 shows that fix-inducing changes require more time than non-fix-inducing. However, in the Kword project non-fix-inducing defects required more time than fix-inducing ones. The defect resolving time was quite high in the Writer, Calc and Apache projects. In the Kword, Glib and Gimp projects, fix-inducing defects were resolved quickly.

8.3 VM process workflow and management

Calc and Writer projects use Environmental Information System (EIS) to manage source code branches, called child workspaces (CWS). Due to the usage of EIS and CWS, the attribute *comment* in the CVS logs does not express reasons for changes or related defects. The comment attribute is used to express the identification of CWS. Therefore,

metrics that are related to the defect-initiated changes cannot be extracted from Calc and Writer. [OOo06a, OOo06b, OOo06c]

Table 8.5 presents the number of defect-initiated changes, which is one of the metrics for the VM process workflow and management. However, it should be compared with two metrics for the activity of the process: Fixed defects and Changes of the source code.

Table 8.5. Metrics for defect-initiated changes in the case studies.

Metrics	Apache	Firefox	Gimp	Glib	Kword
Fixed defects	288	2415	1114	390	207
Changes of the source code[2]	2877	2884	7102	1262	606
Defect-initiated changes[3]	276	1762	1413	370	86
Proportion of fixed defects[4]	31 %	12 %	41 %	58 %	56 %
Proportion of defect-initiated changes[5]	10 %	61 %	20 %	29 %	14 %

1-2) see Table 6.1
3-5) see Table 6.7

As we can see from the table, the number of defect-initiated changes is much lower than the number of changes of the source code. The low proportion of the defect-initiated changes per fixed defects reflects he quality of the comments of the software modification. The proportion of defect-initiated changes was very low in the majority of projects. The Mozilla Firefox project was an exception, with quite a high proportion.

Low proportions in both metrics indicate a lack of efficiency and of communication in the maintenance process. For example, if a comment on the change of the source code does not show that the defect was fixed, other developers cannot know that the defect was resolved. The proportion of defect-initiated changes should be evaluated in shorter periods. Figure 8.1 presents monthly proportions of the defect-initiated changes in the projects.

The figure shows that in most of the case studies the proportion of defect-initiated changes is decreasing. However, in the Mozilla Firefox project, the proportion was increasing, and reached 80 percent. An increasing proportion could be understood as increasing role of DMS.

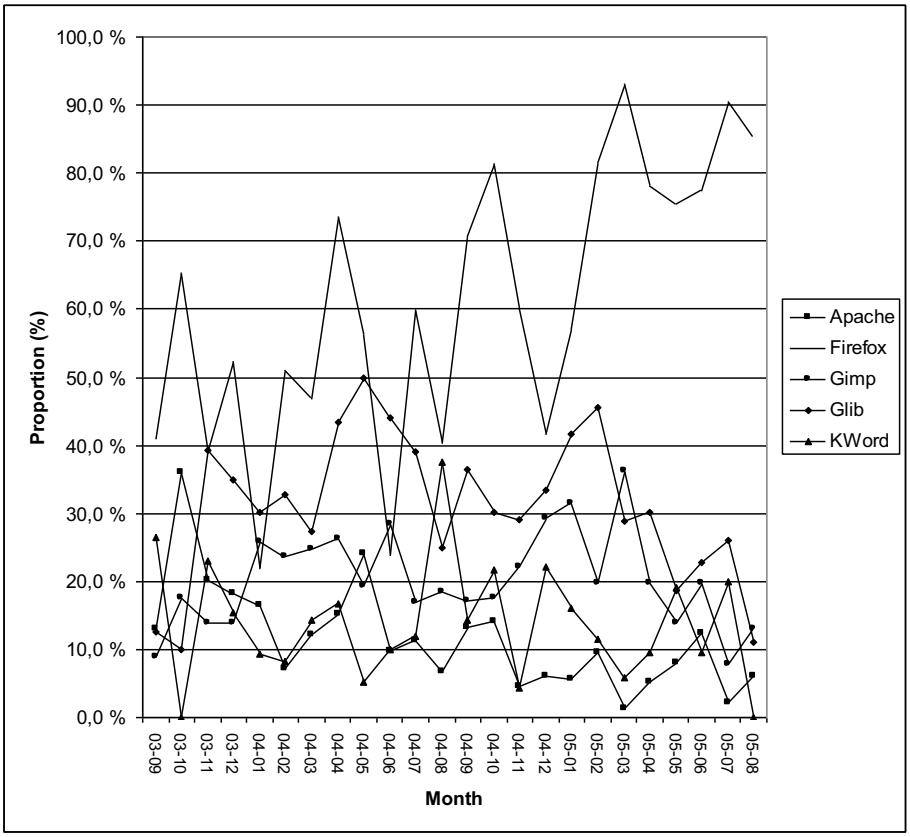


Figure 8.1. Proportion of defect-initiated changes in the case studies.

Instead of analysing all developers separately, we divided authors into three groups according to their activity. According to [CLM03a, CLM03b], most OSSPs have only a few core developers who make most of the changes. To cover the majority of changes with a few developers per project,who submitted more than ten changes per month, . which is more than 240 changes in the two-year evaluation period. They represented over 75 % of all changes. The second group consisted of authors who had submitted more than 24 modifications but fewer than 240, which is fewer than 10 per month. They represent 20 % of all changes. In the third group, which the users included authors, have submitted less than 24 modifications, which is less than one per month. Their proportion of the changes was less than 5 %. The proportions of changes in the group are presented in Figure 8.2.

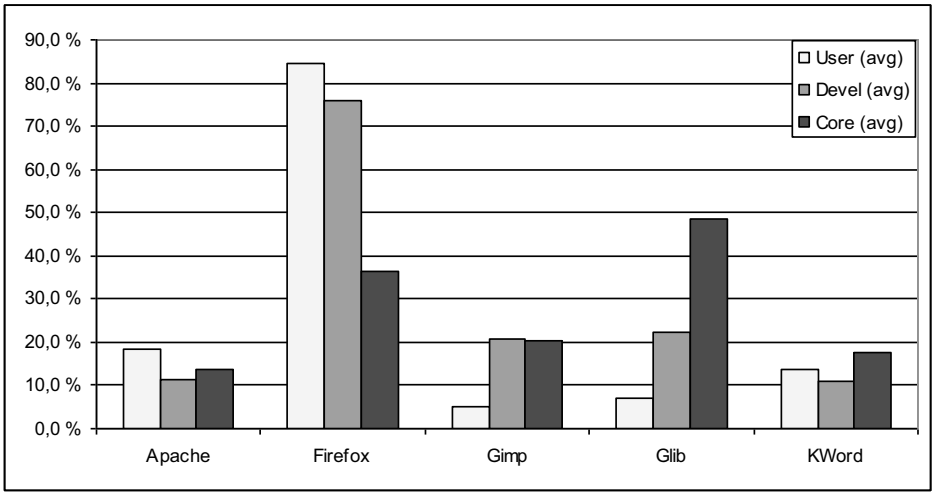


Figure 8.2. Average proportions of defect-initiated changes in different groups.

In the Glib and Gimp projects, active developers (`Devel` and `Core`) had a higher proportion of defect-initiated changes. In contrast, in the Firefox project less active developers had a higher proportion of defect-initiated changes. In the other projects, there were no significant differences between developer groups.

9 CONCLUSIONS AND FUTURE WORK

Open Source Software is quite a new phenomenon and there is ongoing research in several fields. However, no information about the development and maintenance peculiarities of OSS has been available to researchers. The purpose of this thesis was to study maintenance processes in OSSPs. In this section we present a brief summary of the contribution of the thesis (Section 9.1) and future work (Section 9.2).

9.1 Contribution of the thesis

The main contribution of this thesis is the approach for the evaluation of Open Source Software maintenance processes and the tool, Remote analysis System for Open Source Software (RaSOSS). RaSOSS retrieves and analyses data from public DMSs and VMSs, which are used in OSSPs. A more detailed description of the contribution is presented in Figure 9.1. The figure presents the relations between research objectives, topics of the thesis, original publications and their contributions.

Our first research objective was to establish a *framework for Open Source Software projects*. In Section 2 and Paper I we present three major characteristics of OSSP - an information product, community and services. Regarding the information product, we present software and license types. Regarding the community, we present community and development models, and the roles of DMSs and VMSs. Finally, regarding services, we present those which users may require but OSSPs do not necessarily provide.

Our second objective was to establish a *framework for Open Source Software maintenance process*. Before presenting the framework, we present, in Section 3 and Paper II, the standard model for the maintenance process from IEEE and ISO/IEC standards. Thereafter, we studied the maintenance activities of OSSPs and placed these activities in the framework. When we compared our OSS maintenance process framework with the standard process model, we found that both processes have similar activities. However, we also found that the OSS maintenance process does not have retirement activities, which are found in the standard model to ensure that information about the software is still available after the software trade-off.

Our third objective was to create an *evaluation framework for Open Source Software maintenance* and a tool that can be used to analyze Open Source Software maintenance processes. We present in Sections 4 and 5 defect management systems and version management systems and example systems that are used in OSSPs.

We started by studying defect management in OSSPs and found that most of the reported defects did not cause changes to software because many of the reported defects were duplicates or invalidly formed (Paper III). Furthermore, we found that defect reports were rarely marked as assigned, although defect management systems and project guidelines requiring this (Paper IV). One explanation for this may be the style of change man-

agement processes: the changes are reviewed and accepted after their implementation. Even traditionally, changes are accepted before implementation. We found that defect management systems make it possible to study OSS maintenance processes remotely.

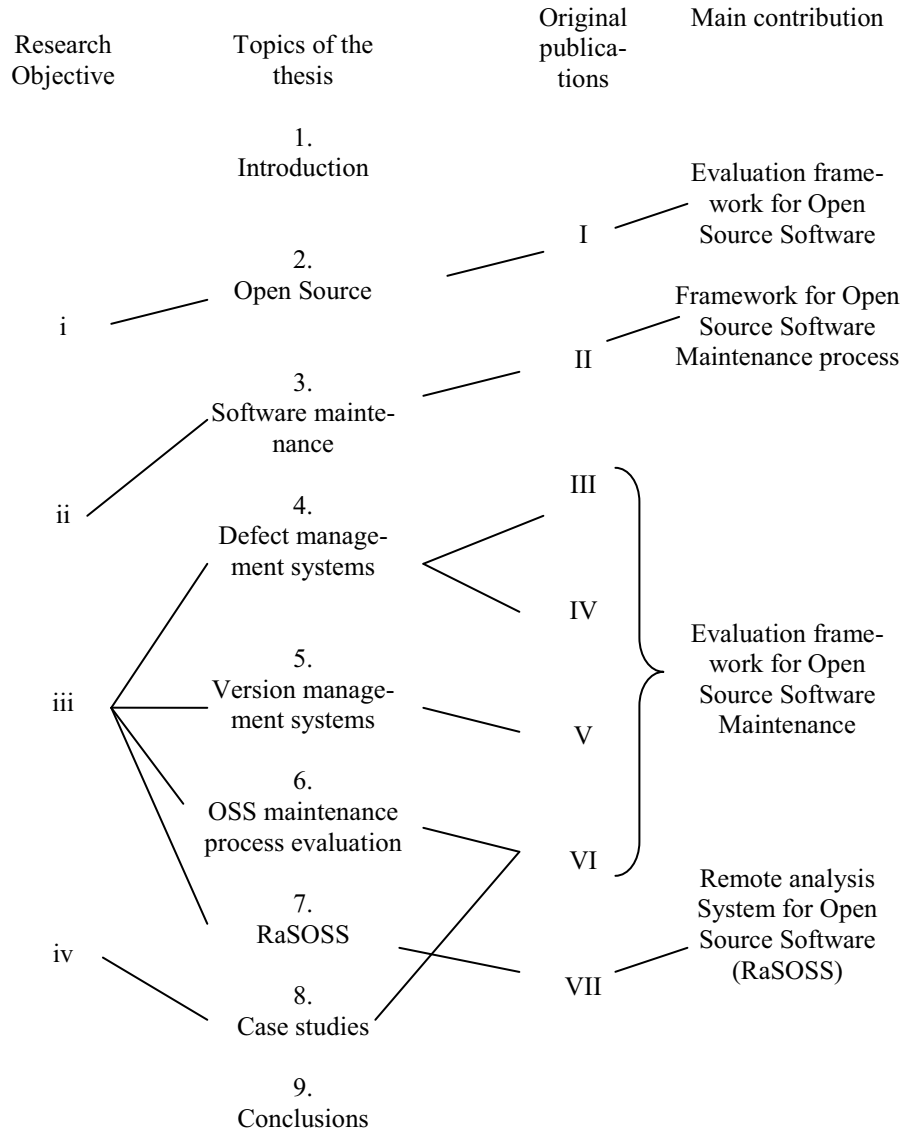


Figure 9.1. The relation between research objectives, topics of the thesis, original publications and main contributions of the papers.

After studying OSSPs through defect management systems, we continued by studying version management and their systems. After we had presented a general overview of version management and example systems, we studied the relation between changes and defect reports. The information about the changes makes it possible to couple the changes to the defects if a defect has caused a change. We evaluated this approach by using case studies in Paper V, and found that many of the changes were not initiated by defects. However, we also found that the proportion of defect-initiated changes may reflect the role of defect management in the OSS maintenance process.

By gathering measurements from studies of our earlier Papers I, II, III, IV and V, we formed an *evaluation framework for Open Source Software maintenance,* which is presented in Section 6 and in Paper VI. However, when studying defect management and version management, we found that the amount of information is too large to be gathered, processed and analyzed manually. Therefore, we created RaSOSS for data gathering, processing and analysis. This tool allows users to analyze OSSPs by just entering the addresses of defect management and version management systems. The tool is presented in Section 7 and Paper VII.

Our fourth objective was to analyze Open Source Software maintenance processes. During the research, we analyzed seven OSSPs using our tool, RaSOSS. The results of the analysis are presented in Section 8 and in Paper VI concurrently with the framework for OSS maintenance process evaluation. We found that our framework is suitable for the evaluation of maintenance processes and it provides information about maintenance processes. However, the results and observations may not be generalizable to all OSSPs because of the low number of case studies.

The main conclusion from the case studies is that, by using the framework, they revealed that maintenance processes seem to be uncontrolled and untraceable. The main finding of the case studies was that the framework and tool were suitable for evaluation. In addition, we found that the activity of the maintenance process was different in some case studies such as Apache, Glib and Gimp. In these projects, defect reports were resolved and fixed seasonally, and most of the time they were not resolved according to DMSs. In the other case studies, rates were more constant and there were no such seasonal trends. Another finding was that the number of source code changes was much higher than the number of fixed defects or even defect reports. This can be the result of inefficient usage of DMS, where defects are not marked as fixed or resolved. Alternatively, it can also be the consequence of multiple changes per resolved defect. When the relationship between the changes and defect reports were studied, we found that the majority of the changes do not seem to be related to defect reports, which also suggests that DMS was used inefficiently. Therefore, we analyzed the workflows of the maintenance process and found that actual workflows are much simpler than project documents or DMSs suggest. The most common workflow for a defect was that it was opened and then resolved, while documents and DMSs suggest that the defects were first verified, assigned and then resolved. All the projects had omitted these steps according to DMSs.

9.2 Future work

There are still open issues in the OSS maintenance processes which would be interesting to study further. Let us propose a few examples. Our information retrieval used earlier studies and data from public DMSs and VMSs, but OSSPs also use mailing lists, newsgroups and discussion boards. The evaluation of these data sources could provide useful knowledge since possibly most of the communication between developers goes through mailing lists, at least in a few cases such as Apache. However, Sandusky et al. [SaG05] have reported that assignment was not a common discussion topic in the Mozilla project, since assignment was related to only seven percent of the topics.

In addition, our study provided an evaluation of OSS maintenance processes from the viewpoint of users and public data. We did not interview developers from OSSPs, but their knowledge could be used to produce more understanding about maintenance processes. By combining these approaches, actual maintenance processes would be well analysable.

As we stated in Sections 4 and 5, a large number of different defect management and version management systems are used in OSSPs. Since our RaSOSS currently supports only a few major VMSs and DMSs, enhancements such as additional data retrieval modules would make it possible to analyze other projects.

Another interesting aspect of the OSS maintenance process is these management systems. One topic for further research could be their evaluation in details. As far as we know, the systems that we worked with did not have major differences in stored information.

Since we studied defect reports that were submitted manually, developers and projects have been developing automated systems for crash reporting. Such systems have already been built in Microsoft Windows XP and Mozilla Firefox 2.0. However, one problem with these systems is that they report only if software crashes. They are not able to report defects if software does not crash or if enhancements are required. But these systems may cause changes in the number and type of defect reports.

BIBLIOGRAPHY

[ADH04] Antoniol G., Di Penta M., Hall H., Pinzger M.: Towards the Integration of CVS Repositories, Bug Reporting and Source Code Meta-Models. In the 2nd Workshop on Software Evolution through Transformations: Model Based vs. Implementation-level Solutions, Rome, Italy, 2004. Electronic Notes in Theoretical Computer Science, Elsevier.

[Aeg06] Aegis: Aegis 4.22. http://aegis.sourceforge.net. (27.7.2006)

[Apa06] Apache Foundation: Welcome - The Apache HTTPD Server Project. http://httpd.apache.org. (27.7.2006)

[APC98] Appleton B., Perczuk S., Cabrera R., Orenstein R.: Streamed Lines: Branching Patterns for Parallel Software Development. http://www.cmcrossroads.com/bradapp/acme/branching/. (13.6.2006)

[Arc06] Free Software Foundation: GNU Arch - GNU Project. http://www.gnu.org/software/gnu-arch. (27.7.2006)

[ARG06] SourceForge.net: SourceForge.net: Argus Issue Tracking System. http://www.sourceforge.net/projects/argus-tracker. (27.7.2006)

[AsB02] Asklund U., Bendix L.: A study of Configuration Management in Open Source Software Projects. IEEE Proc-Software. Vol. 149, No. 1, February 2002.

[Ask02] Asklund U.: Configuration Management for Distributed Development in an Integrated Environment. Doctoral Dissertation, Lund University, Faculty of Technology, Sweden. December 2002. ISBN 91-628-5470-4.

[Baz06] Canonical Ltd: Welcome - Bazaar NG. http.//bazaar-vcs.org. (27.7.2006)

[BCR94] Basili V., Caldiera G., Rombach H.: The Goal Question Metric Approach. Encyclopedia of Software Engineering, 1994.

[Ben00] Bennett K.: Software Maintenance - A Tutorial. Software Engineering (Ed. R Thayer). IEEE Computer Society 2000, pp. 289-304.

[BeR00] Bennett K., Rajlich T.: Software Maintenance and Evolution: A Road-In: Proceedings of the Conference on The Future of Software Engineering. pp. 73-87. ACM Press New York, NY, USA, 2000.

[Bes06] Best Practical Solutions: RT: Request Tracker. http://www.bestpractical.com/rt. (27.7.2006)

[BPR06] BRP Online Learning Center: Reengineering and Process Metrics. ProSci. http://www.prosci.com/metrics.htm. (1.8.2006)

[CLM02] Capiluppi A., Lago P., Morisio M.: Characterizing the OSS Process. Proceedings of the International Conference on Software Engineering, 2nd Workshop on Open Source Software Engineering, Orlando, Florida, May 2002.

[CLM03a] Capiluppi, A., Lago, P., Morisio, M., Characteristics of open source Projects. Proceedings of the Seventh European Conference on Software Maintenance and Reengineering, 2003, pp. 317-327, March 2003.

[CLM03b] Capiluppi, A.; Lago, P.; Morisio, M.: Evidences in the evolution of OS projects through changelog analysis. Proceedings of the 3rd Workshop on Open Source Software Engineering, ICSE'03, Portland, Oregon, USA, 2003. pp. 19-24.

[CMB06] CM Crossroads: CM Crossroads. http.//www.cmcrossroads.com. (15.7.2006)

[CoW95] Cook J., Wolf A.: Automating Process Discovery through Event-data Analysis. Proceedings of the 17th International Conference on Software Engineering (ICSE'95), 1995. pp. 73-82.

[CuS97] Cusumano M., Selby R.: How Microsoft Builds Software. Communications of the ACM, Vol. 40, No. 6, pp. 53-61, ACM Press New York, NY, USA, 1997.

[CVA06] CVSAnalY. http://cvsanaly.tigris.org. (27.7.2006)

[CVS05] CVSHome: Domain Home Page. Collabnet Inc, 2005. http://www.cvshome.org/. (27.7.2006)

[Dar06] Roundy D.: darcs. http://www.darcs.net. (27.7.2006)

[DOS99] DiBona C., Ockman S., Stone M.: OPENSOURCES, Voices from the Open Source Revolution. O'Reilly & Associates, Sebastopol CA, USA, 1999.

[Ere03] Erenkranz, J.: Release Management Within Open Source Projects. Proceedings of the International Conference on Software Engineering, 3rd Workshop on Open Source Software Engineering, Portland, Oregon, February 2003.

[Eve06] Eventum: Main Page - Eventum. http://eventum.mysql.org. (23.5.2006)

[FeP97] Fenton N., Pfleeger S.: Software metrics: A Rigorous and Practical Approach. PWS Publishing Co. Boston, MA, USA, 1997.

[Fin06] Finlex: Ajantasainen Lainsäädäntö - 8.7.1961/404. Tekijänoikeuslaki. 8.7.1961/404. http://www.finlex.fi/fi/laki/ajantasa/1961/19610404 (28.11.2006)

[Fog90] Fogel K.: Open Source Development with CVS. Coriolis Group, Arizona, 1999.

[FSF06a] Free Software Foundation: Free Software Definition - GNU Project. http://www.gnu.org/philosophy/free-sw.html. (12.3.2006)

[FSF06b] Free Software Foundation: GNATS - GNU Bug Tracking Software. http.//www.gnu.org/software/gnats. (24.7.2006)

[Ger04] German D.: Mining CVS Repositories, the softChange Experience. First International Workshop in Mining Software Repositories, 2004.

[GHJ04] German D., Hindle A., Jordan N.: Visualizing the Evolution of Software Using softChange. In Software Engineering Knowledge Engineering (SEKE'04), 2004.

[Gia05] Giacomo, P.: COTS and Open Source Software Components: Are they Really Different on the Battlefield. Proceedings of the 4th International Conference on COTS-Based Software Systems, Bilbao, Spain, February 2005.

[Gim06] Gimp.org: GIMP - The GNU Image Manipulation Program. http://www.gimp.org. (21.4.2006)

[Git06] Git: Git - Fast Version Control System. http://git.or.cz. (24.7.2006)

[Gli06] GNU Project: GTK+ - The Gimp Toolkit. http://www.gtk.org. (24.5.2006)

[HNH03] Hertel G., Niedman S., Herrmann S.: PR Special Issue: Motivation of Software Developers in Open Source Projects: An Internet-based Survey of Contributors to the Linux Kernel. http://opensource.mit.edu/papers/rp-hertelniednerherrmann.pdf. (7.3.2003)

[IEE98] IEEE: IEEE Standard for Software Maintenance, IEEE Std 1219-1998. The Institute of Electrical and Electronics Engineering, Inc. 1998.

[IEE04] IEEE: Guide to the Software Engineering Body of Knowledge (SWE-BOK). IEEE Computer Society 2004, Los Alamitos, California.

[Ins06] Institut für Rechtsfragen der Frein und Open Source Software: License Center. Institut für Rechtsfragen der Frein und Open Source Software, http://www.ifross.de/ifross_html/lizenzcenter-en.html. (20.8.2006)

[ISO99] ISO/IEC: ISO/IEC 14764:1999: Information Technology – Software Maintenance. ISO/IEC 1999.

[ISO00] ISO: Quality Management Systems – Fundamentals and Vocabulary (ISO 9000:2000). ISO 2000.

[ISO02a] ISO/IEC: ISO/IEC 12207:1995/Amd 2002: Software Engineering: Software Life Cycle Processes. ISO/IEC 2002.

[ISO02b] ISO/IEC: ISO/IEC 15288:2002: Software Engineering: Software Life Cycle Processes. ISO/IEC 2002.

[ISS06] IssueTrackerProduct: IssueTrackerProduct (User Friendly Issue - Bug-Tracking Open Source Web Application for Zope). http://www.issuetrackerproduct.com. (24.7.2006)

[KaM06] Kajko-Mattsson M.: Applicability of IEEE 1219 within Corrective Maintenance. The International Conference on Software Engineering Advances (ICSEA'06). October 2006, Tahiti, French Polynesia. (In Press). IEEE Press.

[KDE06] KDE: KDE Bug Tracking System. http://bugs.kde.org. (20.08.2006)

[KiW06] Kim S., Whitehead E.: How Long Did it Take to Fix Bugs? Proceedings of the 2006 International Workshop on Mining Software Repositories (Shanghai, China, May 2006). MSR '06. pp. 173-174. ACM Press, New York, NY 2006.

[KLH06] Koponen T., Lintula H., Hotti V.: Defect reports in Open Source Software Maintenance Process - OpenOffice.org Case Study. In Proceedings of the 10th IASTED SEA, Nov 2006. ACTA Press (In press).

[KOf06] The KOffice Project: KWord. http://www.koffice.org/Kword. (23.07.2006)

[KoH04] Koponen T, Hotti V. Evaluation Framework of Open Source Software. Proceedings of The International Conference on Software Engineering Research and Practice SERP'04, Vol. II., Las Vegas, Nevada, USA, June 21-24, 2004, pp. 897-902. CSREA Press, 2004.

[Kop06a] Koponen T.: RaSOSS - Remote Analysis System for Open Source Software. The International Conference on Software Engineering Advances (ICSEA'06). October 2006, Tahiti, French Polynesia. (In Press). IEEE Press.

[Kop06b] Koponen T.: Evaluation Framework for Open Source Software Maintenance. The International Conference on Software Engineering Advances (ICSEA'06). October 2006, Tahiti, French Polynesia. (In Press). IEEE Press.

[LeT00] Lerner, J., Tirole J.: The Simple Economics of Open Source Software, NBER Working Paper 7600, 2000. http://www.nber.org/papers/w7600

[LeT02] Lerner, J., Tirole J.: Some Simple Economics of Open Source. The Journal of Industrial Economics, Vol. 50, No. 2, June 2002, pp. 197-234.

[Lib06] Libresoft: GlueTheos: http://libresoft.urjc.es/Tools/GlueTheos. (3.7.2006)

[Lon05] Lonchamp J.: Open Source Software Development Process Modelling. (S.T. Acuna, N. Juristo Eds.) Software Process Modelling, Springer, 2005. ISBN: 0-387-24261-9.

[Man06] Mantis.org: Mantis Bug Tracker. http://www.mantisbt.org. (14.7.2006)

[Mas05] Massey B.: Longitudinal Analysis of Long-timescale Open Source Repository Data. Proceedings of the 2005 Workshop on Predictor Models in Software Engineering. St. Louis, Missouri, May 2005. PROMISE '05. ACM Press, New York, NY, pp. 1-5.

[Mer06] Mercurial: Mercurial. http.//www.selenic.com/mercurial. (27.7.2006)

[MFH02] Mockus A., Fielding R., Herbsleb J.: Two Case Studies of Open Source Software Development: Apache and Mozilla. ACM Trans. Software Engineering and Methodology, 11(3), 309-346. ACM Press, 2002.

[Mon06] Monotone: Distributed version control. http://venge.net/monotone. (23.7.2006)

[Moz05] Mozilla.org: Bugzilla. Mozilla.org, 2005. http://www.mozilla.org/bugzilla. (22.2.2005)

[Moz06a] Mozilla: Firefox - Rediscover the Web. http://www.mozilla.org/firefox. (22.6.2006)

[Moz06b] Mozilla: Firefox Downloads Counts. http://feeds.spreadfirefox.com/downloads/firefox.xml. (27.7.2006)

[Net06] Netcraft: Netcraft. http://www.netcraft.com/. (13.5.2006)

[NYN02] Nakakoji K., Yamamoto Y., Nishinaka Y., Kishida K., Ye Y.: Evolution Patterns of Open-Source Software Systems and Communities. In Proceedings of the International Workshop on Principles of Software Evolution (Orlando, Florida, May 2002). IWPSE '02. ACM Press, New York, NY, pp. 76-85.

[OCM06] OpenCM.org: OpenCM Website. http://www.opencm.org. (4.7.2006)

[OGC02a] Office of Government Commerce: Infrastructure Management. IT Infrastructure Library (ITIL). The Stationary Office, Norwich, UK, 2002.

[OGC02b] Office of Government Commerce: Application Management. IT Infrastructure Library (ITIL). The Stationary Office, Norwich, UK, 2002.

[OOo06a] OpenOffice.org: EIS. wiki.services.openoffice.org/wiki/EIS. (27.7.2006)

[OOo06b] OpenOffice.org: CWS. wiki.services.openoffice.org/wiki/CWS. (27.7.2006)

[OOo06c] OpenOffice.org: Tools: CWS Howto. http://tools.openoffice.org/dev_docs/OOo_cws.html. (27.7.2006)

[OOo06d] OpenOffice.org: Spreatsheet Project. http://sc.openoffice.org. (14.5.2006)

[OOo06e] OpenOffice.org: sc: Issue Tracker. http://sc.openoffice.org/servlets/ProjectIssues. (7.5.2006)

[OOo06f] OpenOffice.org: Word Processing Project. http://sw.openoffice.org. (22.5.2006)

[OSI06] Open Source Initiative: The Open Source Definition. Open Source Initiative 2003. http://www.opensource.org/docs/definition.php. (6.8.2006)

[OTR06] OTRS.org: OTRS::Email Management::Trouble Ticket System:: Welcome. http://www.otrs.org. (27.7.2006)

[PHP06] PhpBugTracker: phpBugTracker. http://phpbt.sourceforge.net. (27.7.2006)

[Ray99] Raymond E.: The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. O'Reilly & Associates, Inc. Sebastopol, CA, USA, 1999.

[RCS06] Free Software Foundation: RCS - GNU Project. http://www.gnu.org/softwarercs/rcs.html. (27.7.2006)

[RGG04] Robles G., Gonzalez-Barahona J. Ghosh R.: GlueTheos: Automating the Retrieval and Analysis of Data from Publicly Available Software Repositories. First International Workshop in Mining Software Re-positories (MSR'04), 2004. pp. 28-31.

[RiG03a] Ripoche G., Gasser L.: Distributed Collective Practices and F/OSS Problem Management: Perspective and Methods. In the Proceedings of the Conference on Cooperation, Innovation & Technologies (CITE 2003). 2003.

[RiG03b] Ripoche G., Gasser L.: Scalable Automatic Extraction of Process Models for Understanding F/OSS Bug Repair. (Hamza H. eds.), Proceedings of the 16th International Conference on Software & Systems Engineering and their Applications (ICSSEA-03). December 2003, ACTA press.

[RKG04] Robles G., Koch S., González-Barahona J.: Remote Analysis and Measurement of Libre Software Systems by Means of CVSAnalY tool. Proceedings of the 2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems (RAMSS '04). 2004.

[Ros00] Rosenberg D.: Open Source: The Unauthorized White Papers. IDG Books Worldwide , M&T Books, Forter city, CA, USA, 2000.

[Ros04] Rosen L.: Open Source Licensing : Software Freedom and Intellectual Property Law. Prentice-Hall, USA, 2004.

[Rot02] Rothfuss G.J.: A Framework for Open Source Projects. Master's Thesis in Computer Science. Department of Information Technology, University of Zürich, 2002.

[San01] Sansdred J.: Managing Open Source Projects: A Wiley Tech Brief. John Wiley & Sons Inc, NY, USA 2001.

[SaG05] Sandusky R., Gasser L.: Negotiation and the Coordination of Information and Activity in Distributed Software Problem Management. Proceedings of the 2005 International ACM SIGGROUP Conference on Supporting Group Work, Sanibel Island, Florida, USA, November 2005, (GROUP '05). ACM Press, New York, NY, pp. 187-196.

[Sei06] Software Engineering Institute: SEI Open Systems Glossary. http://www.sei.cmu.edu/opensystems/glossary.html. (9.6.2006)

[Sof06] SoftChange: SoftChange. http://sourcechange.sourceforge.net. (27.7.2006)

[SuW06] Sunghun K., Whitehead J.: How Long Did it Take to Fix Bugs? MSR '06: Proceedings of the 2006 International Workshop on Mining Software Repositories, Shanghai, China, 2006. pp. 173-174. ACM Press, New York, NY, USA, 2006.

[SVK06] svk.elixus.org: The SVK Version Control System. http://svk.elixus.org. (27.7.2006)

[SZZ05] Sliwerski J., Zimmermann T., Zeller A.: When Do Changes Induce Dixes? In MSR '05: Proceedings of The 2005 International Workshop on Mining Software Repositories. ACM Press, New York, USA, 2005.

[TaG03] Takang A., Grubb P.: Software Maintenance: Concepts and Practice. World Scientific Publishing Company, 2003.

[Tig06a] Tigris.org: Scarab.tigris.org. http://scarab.tigris.org. (27.7.2006)

[Tig06b] Tigris.org: Subversion.tigris.org. http//subversion.tigris.org. (22.11.2006)

[Tra06] SourceForge: E03. Tracker (en). http://sourceforge.net/docman/display_doc.php?docid=24202&group_id=1. (20.07.2006)

[VAS06] VASoftware: SourceForge Product Introduction. http://www.vasoftware.com/sourceforge. (27.7.2006)

[Ves06] Vestasys: Vesta Configuration Management System. http://www.vestasys.org. (20.07.2006)

APPENDIX I: Trend metrics of the case studies

Figures 1 to 7 present monthly statistics of the case studies.

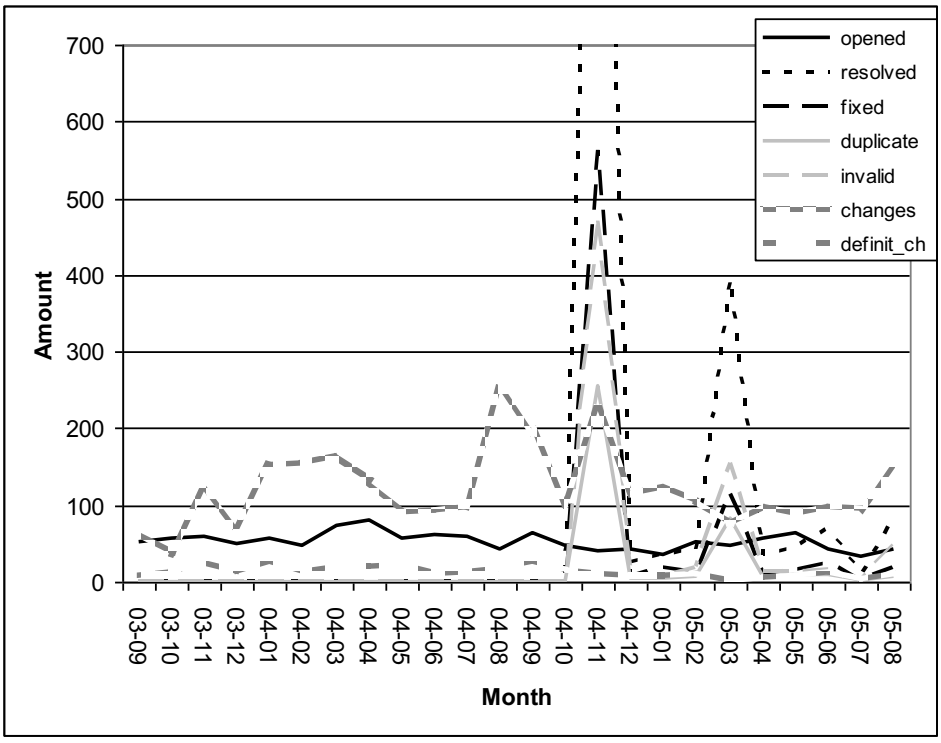


Figure 1. Monthly statistics of the Apache project.

In the Apache project (Figure 1), even though defects were reported between September 2003 and October 200, not a single defect was resolved. However, nearly 1500 defects were resolved during November 2004. The same phenomenon also occurred in March 2005. Otherwise, the monthly number of defects and changes was quite constant.

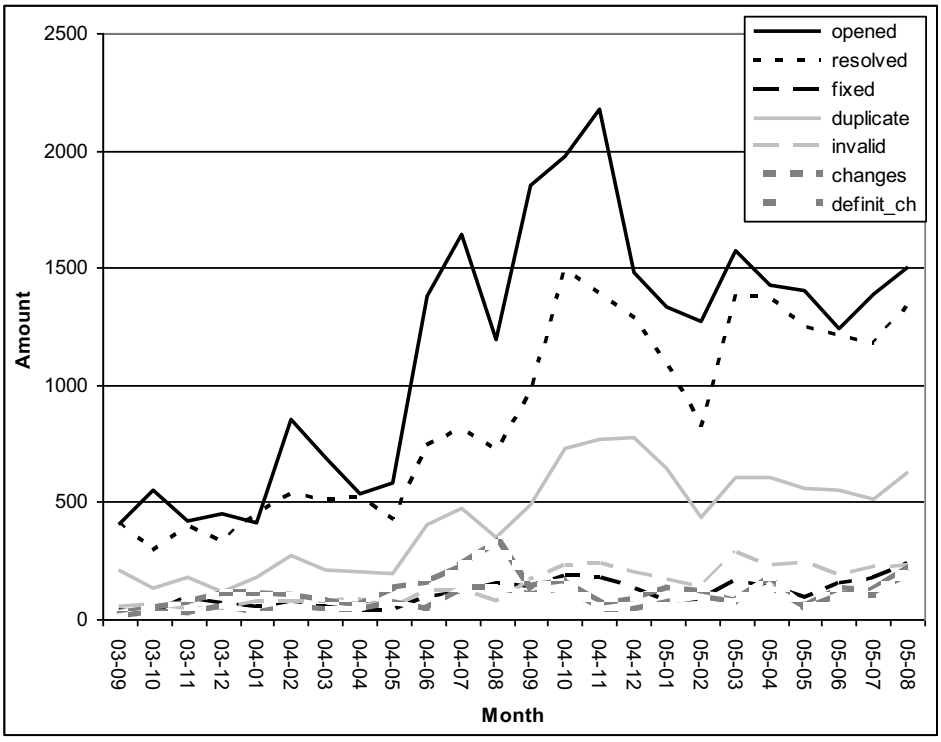


Figure 2. Monthly statistics of the Mozilla Firefox project.

In the Mozilla Firefox project (Figure 2), the number of opened and resolved defects increased and stabilized at a level of over a thousand per month.

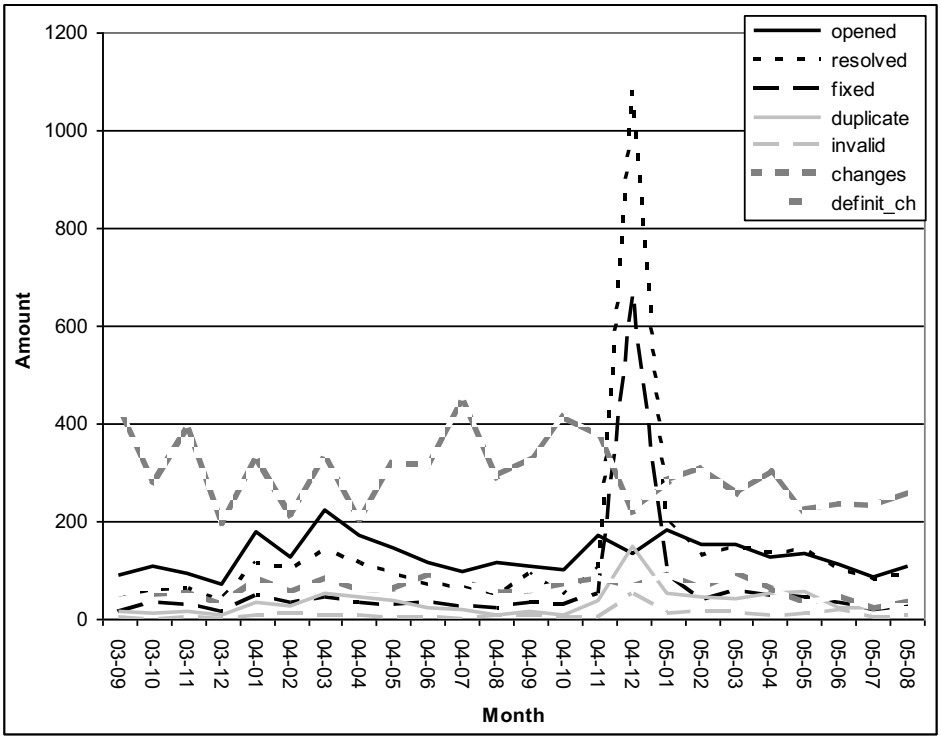


Figure 3. Monthly statistics of the Gimp project.

In the Gimp project (Figure 3), defect reporting and resolving was quite constant during the evaluation period but there were over a thousand resolved defects in December 2004. The overall number of source code modifications decreased slightly.

Because Gimp has plug-ins, modifications to the main product are not always needed. Developers add new functionalities, such as graphical filters, as plug-ins.

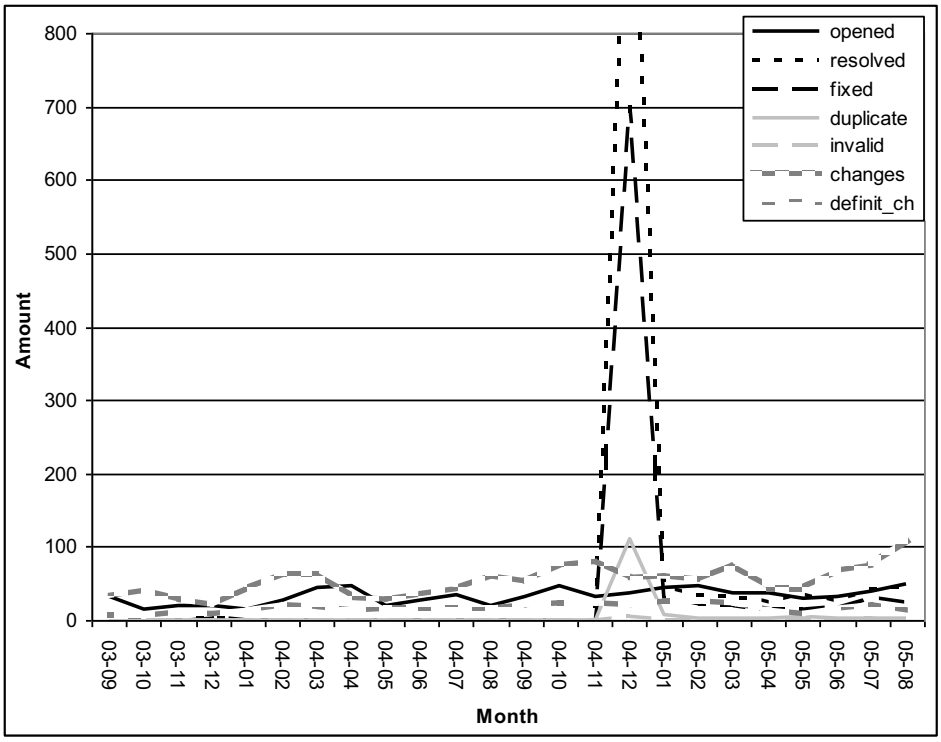


Figure 4. Monthly statistics of the Glib project.

In the Glib project (Figure 4), defect reporting and resolving were quite constant during the evaluation period but there were over one thousand resolved defects in December 2004. However, the number of source code modifications increased since developers were integrating new functionalities and adding support for new platforms.

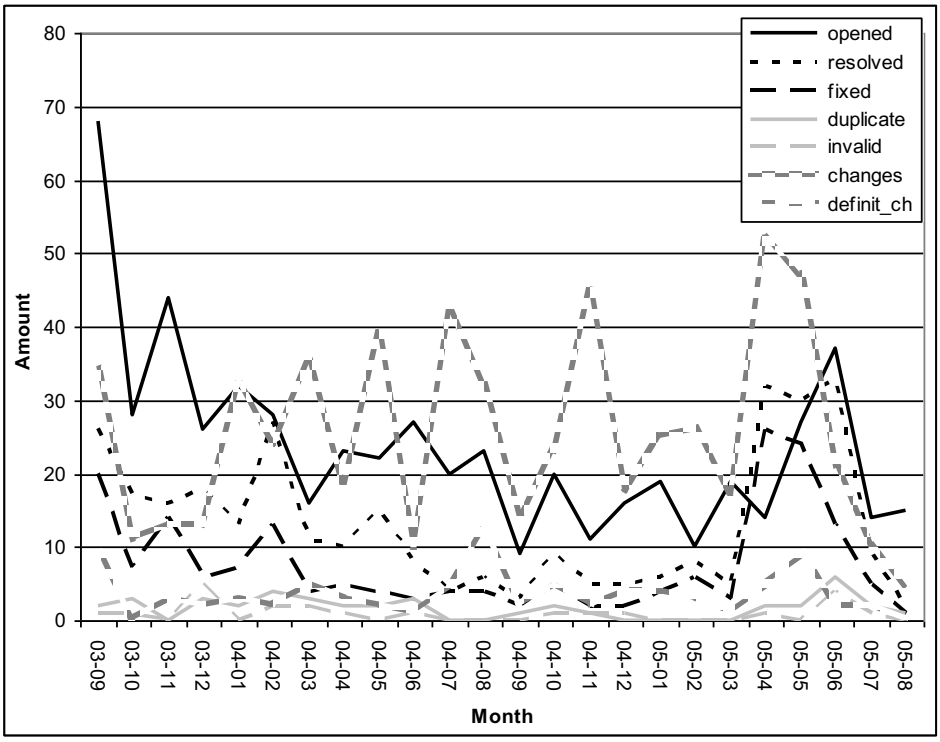


Figure 5. Monthly statistics of the Kword project.

In the Kword project (Figure 5), the number of defects varied during the evaluation period. The figure shows that development is active in the Kword project.

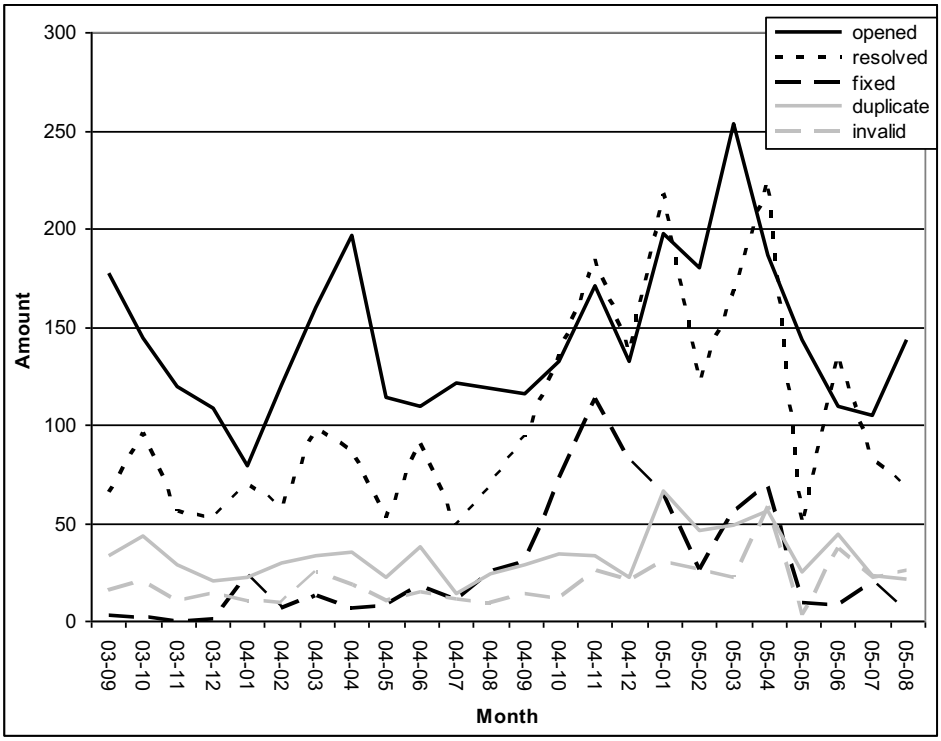


Figure 6. Monthly statistics of the Calc project.

In the Calc project (Figure 6), defect reporting and resolving was quite constant during the evaluation period. However, the source code was heavily modified between May 2004 and November 2004. At the end of the evaluation period, the number of source code modifications decreased to fewer than one hundred per month.

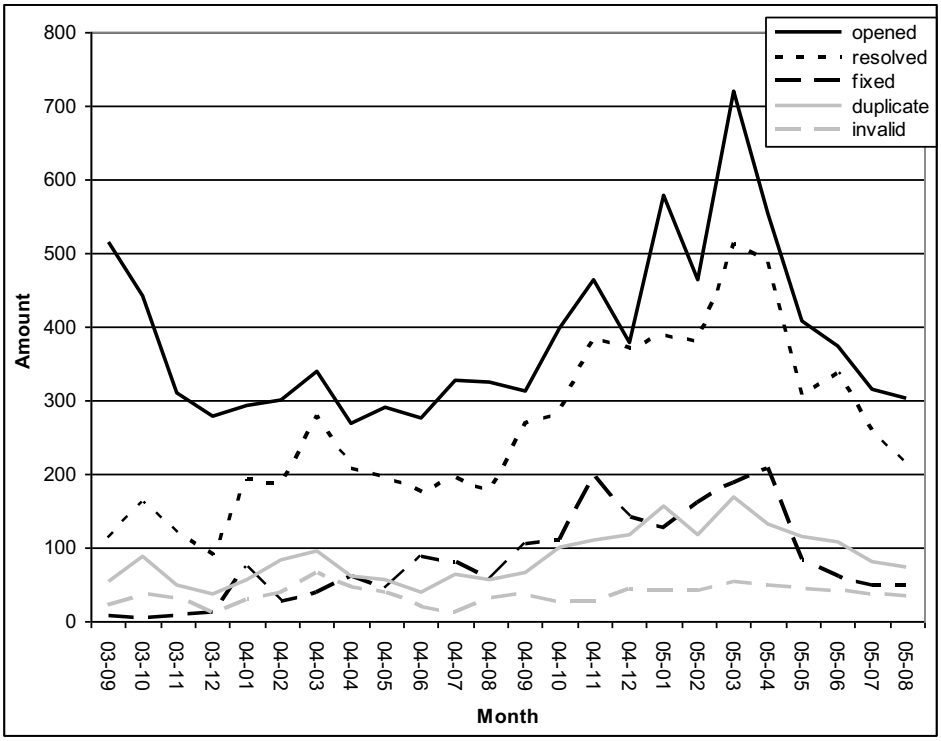


Figure 7. Monthly statistics of the Writer project.

In the Writer project (Figure 7), defect reporting and resolving was quite constant during the evaluation period. However, the source code was heavily modified between May 2004 and November 2004. At the end of the evaluation period, the number of source code changes decreased to fewer than one hundred per month.

Kuopio University Publications H. Business and Information technology

H 1. Pasanen, Mika. In Search of Factors Affecting SME Performance: The Case of Eastern Finland.
2003. 338 p. Acad. Diss.

H 2. Leinonen, Paula. Automation of document structure transformations.
2004. 68 p. Acad. Diss.

H 3. Kaikkonen, Virpi. Essays on the entrepreneurial process in rural micro firms.
2005. 130 p. Acad. Diss.

H 4. Honkanen, Risto. Towards Optical Communication in Parallel Computing.
2006. 80 p. Acad. Diss.

H 5. Laukkanen, Tommi. Consumer Value Drivers in Electronic Banking.
2006. 115 p. Acad. Diss.

H 6. Mykkänen, Juha. Specification of reusable integration solutions in health information systems.
2006. 88 p. Acad. Diss.

H 7. Huovinen, Jari. Tapayrittäjyys – tilannetekijät toiminnan taustalla ja yrittäjäkokemuksen merkitys yritystoiminnassa.
2007. 277 p. Acad. Diss.

H 8. Päivinen, Niina. Scale-free Clustering: A Quest for the Hidden Knowledge.
2007. 57 p. Acad. Diss.