

# Are the Changes Induced by the Defect Reports in the Open Source Software Maintenance?

Timo Koponen, Heli Lintula

Department of Computer Science

University of Kuopio,

Finland

timo.koponen@uku.fi, heli.lintula@uku.fi

## Abstract

In this paper we study defect management and version management systems that are used in many Open Source software projects to manage development and maintenance processes. Usually, those systems are not integrated together even there is a conceptual relation between stored information.

Our purpose was to create an approach that could be used to analyze relations between changes and defects. To evaluate our approach we analyzed two case studies, Apache HTTP Server and Mozilla Firefox that are well-known examples of Open Source software.

Consequently, we found that only a small percentage of changes in the source code of the Apache were initiated by defect reports. However, in Mozilla over 60 percent of changes were initiated by defect reports. Furthermore, we found that developers which had made fewer changes were more likely to do changes that were initiated by defect.

Keywords: Maintenance, Open Source.

## 1 Introduction

Open Source users are encouraged to report defects, which they have with software, and request enhancements. Therefore, users need to have a channel to communicate with the developers. Traditionally Open Source users have reported defects by e-mail, newsgroups or mailing lists. However, when the user group is large enough, those channels are not sufficient. Therefore, many projects use a dedicated system for the defect reporting and management such as Bugzilla [2].

The defect management system provides flexible possibilities to track, control, and assign defects. *Defect reports* are representations of the defects in the defect management system. A defect report contains a detailed description of the defect. It also contains the state and outcome when the defect is resolved. [10]

Many of the Open Source projects have also a version management system that stores all versions

of the source code. Differences between the versions of software are stored as changes. A *change* contains an identification of the author, differences between files and a short description.

These systems are connected through users and developers. Figure 1 presents the relationship between defect management and version management systems.

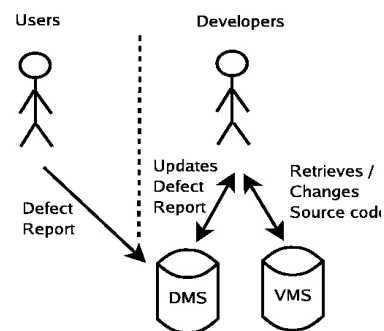


Figure 1: The relation between the defect management (DM) and the version management (VM)

The users communicate with the developers through defect management systems. The users may report defects, check the status of the defect, or browse defects reports to ensure that similar defects have not been reported. The developers retrieve defect reports from the defect management system. Besides that they acquire source codes from the version management system. Then they fix defects or implement enhancements that were reported in defect reports. Later, when the changes are done, source code is updated to the version management system and the status and the resolution of the defect is updated into the defect management system [4, 9].

To study the reliability of the connection we formed three research questions: How to determine which changes were defect initiated. What proportion of the changes were defect initiated. Are the results similar in the both case studies and are the proportions affected by the activity of the developers.

The rest of the paper is organized in the following way; first Section 2 will present the data retrieval

and the classification methods. Then Section 3 will introduce the case studies that were used to study proportions of defect initiated changes. Furthermore, it will present the number of analysed defect reports, all changes and changes that are linked to defect reports. Section 4 will analyse the results of the case studies. Consequently, it will analyse similarities and differences of the case studies and their implications. Finally, Section 5 will introduce related work and Section 6 will conclude work and present future work on the field.

## 2 Method

To retrieve defects and changes of the source code we created software. The software had three main parts that were the defect data extractor, the version history extractor, and the analysis module.

The defect data extractor first retrieved a list of defects that were reported to the system. Thereafter, it collected detailed information of every defect and changes that were made to defects. Our defect data extractor supports Bugzilla because it is used in majority of Open Source projects as a defect management system.

The version history extractor retrieved the source code of the software and then it retrieved the change history of source code files. After the data retrieval both modules exported data to database where the analysis module could classify defects and changes. Our version history extractor supports Concurrent Versions System (CVS) and SubVersion (SVN) because they are two most popular Open Source version management systems.

Table 1: Outcomes of the defect.

Resolution	Explanation
(empty) or Not resolved	The defect does not have resolution yet
Fixed	The defect is fixed and changes to the source code are imported
Works for me	The defect does not occur in other users' system
Wont fix	The defect is not a fault, real problem or it is a feature
Invalid	The defect report is invalid; maybe information is missing
Duplicate	The defect report is duplicate to another

Next Subsection 2.1 will present the classification of defects; and Subsection 2.2 will present the description and classification of the changes.

### 2.1 Defects

Reported defects are, at first, without resolutions. When the defect is resolved it will be closed. However, earlier studies have shown that most of the resolved defects do not cause changes to software [3]. Therefore, an attribute for the *resolution* is needed to express the outcome of the defect. Table 1 describes resolutions that are possible in the Bugzilla system.

In this study, we were interested in defects which resolution was fixed. Those defects are called as *fixed defects*. These fixed defects should induce changes in the source code.

### 2.2 Source Code Changes

For the source code management there are two most popular Open Source software; Concurrent Versions System (CVS) and SubVersion (SVN). They have been created for similar purposes but they have some differences. The major difference from our viewpoint was that in SVN changes are atomic. One change can include changes to multiple files. Meanwhile, in CVS there are no similar atomic changes. A change that includes changes to multiple files is presented as series of changes. However, series of changes in CVS system can be grouped and understood as one atomic change by using author and time information.

Description of change data CVS system is presented in Example 1.

```
revision 1.10
date: 2004/04/18 14:18:14; author:
xxx\%domain.com; state: Exp; lines: +1 -1
Fixed Bug 1234 - Changed variable type from int to
long.
```

The example showed that CVS logs date, time, the author (the developer) and the comment of the change. Furthermore, similar data is logged in SVN. The format of the SVN data is similar to CVS despite that it lists changed files.

In the ideal case every comment of the change would express shortly the type of the change and express clearly if it is related to the defect report. Figure 2 presents the idea of the linking defect and the change of source code when the defect report (wave box) is expressed in the comment (box)

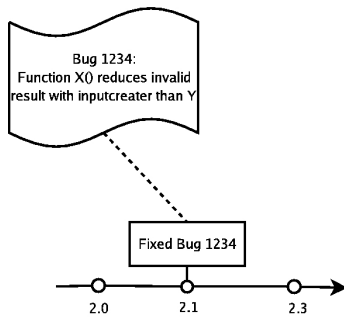


Figure 2: The relation between the Change and the Defect

However, in the real world, different projects and authors include different information in the comments. Therefore, all comments does not express clearly or in the similar format if the change was initiated by defect reports.

To classify changes to defect initiated or non-defect initiated changes all comments of changes had to be analysed. Sliwerski et al. [13] have presented an idea that could be used to find links between changes and defect reports. So, we based our change classification on their idea.

Our change classification proceeds in two steps. In the first step all changes were automatically grouped to seven groups. Grouping was done by searching syntactical identifiers that were:

- Highly probability
  1. PR: [number],
  2. PR# [number],
  3. bug [number].
- Medium probability
  4. a number that has more than 4 digits but less that 8.
  5. a link to Bugzilla or other website.
  6. a word bug, defect, fix or pr.
- Low probability
  7. Others

In the second step, changes from groups 4, 5, 6 and 7 were analysed manually to find defect initiated changes.

Although the candidate search was partly automatic, all changes were manually revised to assure results. We found that over 90 percent of the changes that were automatically classified to groups 1, 2, 3 and 4 were actually defect initiated.

### 3 Case Studies

We analyzed two projects, Apache HTTP Server and Mozilla Firefox. Apache and Mozilla are widely used and their quality is highly appreciated. Therefore they are representative case studies. [8]

To eliminate the age of the project as a confusing factor we selected two year time-period for analysis. We analysed all defects and changes that were reported between September 2003 and September 2005. Table 2 presents the number of analyzed defects and changes in each case.

Table 2: The number of the analysed defects and the changes in the case studies

Case	Changes of the Source code	Defects
Apache	2877	1266
Mozilla	2884	27681

As Table 2 shows, there were more than twice as many changes in the source code than reported defects in Apache. Controversially, in Mozilla, there were ten times more defects than changes of the source code.

Source code changes were done by 47 different developers in Apache (averagely 61 changes per developer). The changes of Mozilla were done by 78 different developers (averagely 36 changes per developer).

However, there were more defect reporters in both cases. In Apache, there were 1027 different persons that had reported defects. Furthermore, in Mozilla there were 16765 different persons that had reported defects. The number of defect reporters was hundred times higher in Mozilla and over ten times higher in Apache.

Table 3: The resolution of the defects in Apache and Mozilla

Resolution	Apache	Mozilla
Duplicate	162	10038
Fixed	288	2414
Invalid	370	3404
Later	5	0
Remind	1	0
WontFix	84	714
WorksForMe	33	3730
Total	943	20300
Not resolved	323	7381
Total	1266	27681

Table 3 presents the resolutions of the defects in the both cases. It shows that Apache had 943 resolved defects and Mozilla had 20300 resolved defects. However, not all resolved defects led to changes. In the case of Apache 288 resolved defects were fixed. It was 31 per cent of the resolved defects. However, in the case of Mozilla, controversially, 2414

resolved defects were fixed. This was less than 12 per cent of the resolved defects. Next, Table 4 presents the number of all changes, defect initiated changes and fixed defects.

Table 4: The number of changes, defect initiated changes and fixed defects in Apache and Mozilla

Case	All Changes	Defect initiated changes	Fixed defects
Apache	2877	276	288
Mozilla	2884	1762	2414

In the case of Mozilla over half of the changes (61%) are linked to defect reports. Interestingly, even there where 2414 fixed defects, there were only 1761 changes that had a link to defects. It raises a question, what happened to the rest of the fixed defects and their changes.

In the case of Apache less than 10 per cent was linked to the defect reports. Therefore, it seems that most of the changes are not initiated by a defect. However, almost all defects that became fixed had a change that was related to a defect.

#### 4 Analysis

The case studies showed that in Apache the majority of changes were not defect initiated. However, in Mozilla half of the changes were defect initiated. To understand the different results we analyzed the dependency of the author activity. Actually, the version management system expresses the author that uploaded the changes – not necessarily the actual developer of the change. Developers and authors can be different persons if the actual developers do not have access to the version management system.

First, we analyzed Apache more carefully. Figure 3a presents the number of the defect initiated changes and the number of the changes per author in the Apache. Furthermore 3b presents the proportion of the defect initiated changes and the number of the changes per author.

The number of the changes per author, on the x-axis, represents the activity of the author. Because of the wide distribution between authors, the scale of the x-axis is logarithmic. The y-axis, which presents the number of the defect initiated changes, is also scaled logarithmically for the same reason.

Figure 3a shows that authors with more changes have also more defect initiated changes. However, Figure 3b shows that author activity does not increase the proportion of defect initiated changes.

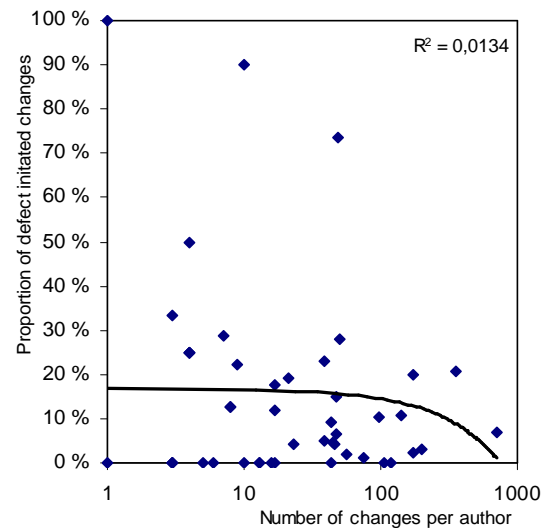
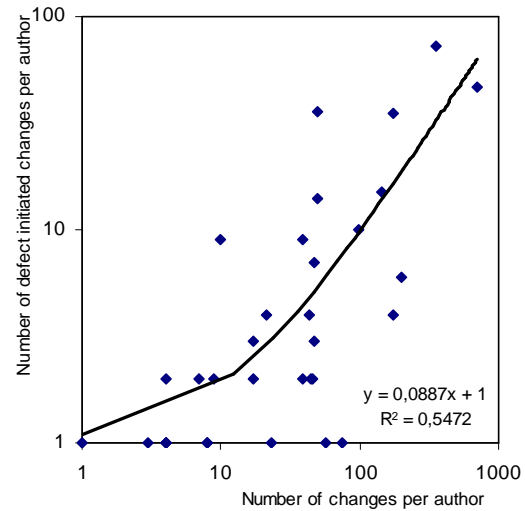


Figure 3: Apache a) The number of changes and defect initiated changes per author. b) The number of changes and proportion of defect initiated changes per author.

Conversely, it seems that more active authors had a smaller proportion of the defect initiated changes.

Even the linear trend in Figure 3b was not well-fitted to data (the correlation coefficient was 0.013), it shows that the trend describes plotted data. The correlation coefficient was poor because of the wide variation in the proportions of the defect initiated changes.

Second, we analyze Mozilla more carefully. Figure 4a presents the number of the defect initiated changes and the changes per author in Mozilla project. Furthermore 4b presents the proportion of

the defect initiated changes and the changes per author.

Figure 4a shows that, also in Mozilla, active authors made more defect initiated changes. However, Figure 4b shows that, also in Mozilla, more active

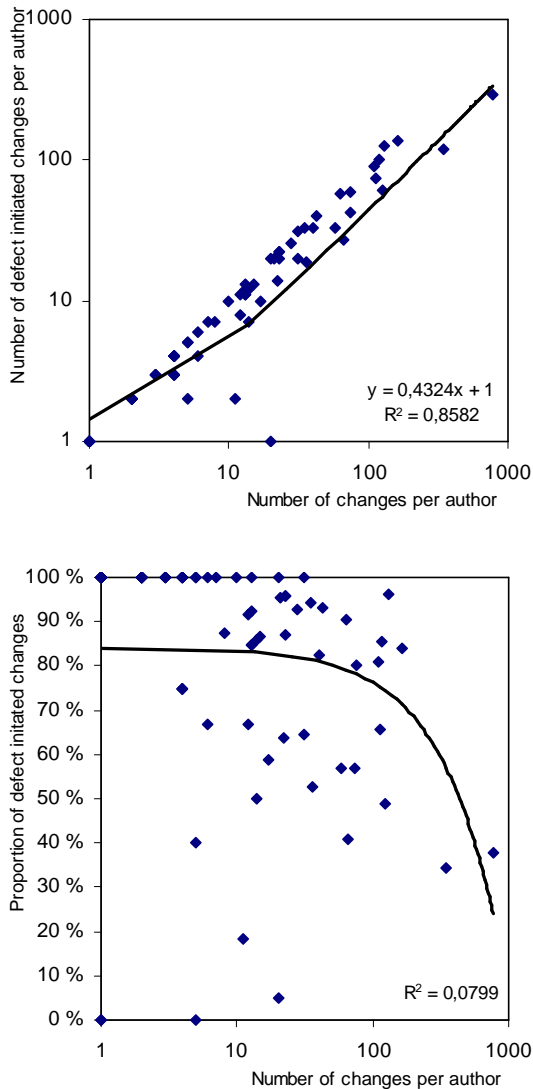


Figure 4: Mozilla. a) The number of changes and defect initiated changes per author. b) The number of changes and proportion of defect initiated changes per author.

authors had a smaller proportion of the defect initiated changes. Like in Figure 3b, the linear trend in Figure 4b was not well fitted (the correlation coefficient was 0.08) for the same reasons.

Even the correlations of the linear trends in Figures 3b and 4b were not statistically significant, they showed that less active authors made defect initiated changes more likely.

Furthermore, in the case of Mozilla, it was strange that all fixed defects didn't lead to changes in the source code. It could be possible that the link between the change and the defect has been forgotten to add in the change data. In addition, it is possible that those defects has been forgotten to close when changes were committed to the version management system. Later, when those defects were closed, their changes were out of the two-year analysis period.

Even the cases had similarities in the effects of the author activity to the proportion of defect initiated changes; the cases had a significantly different total proportion of the defect initiated changes. The difference can be explained with the differences of the cases. Because Apache is server software, which is used by administrators and system operators, there were less defect reports. Maybe developers and users of Apache discuss on the mailing lists and newsgroups. Furthermore, Apache is developed with a smaller developer group and therefore the developers know their system better and make improvements, enhancements and fixes independently.

Conversely, Mozilla is end user software and therefore the number of defect reports is higher. Furthermore, Mozilla has a huge number of users and there are also more developers. However, a larger developer group probably requires stricter control for changes through the defect management system.

## 5 Related Work

To our knowledge, this is the first work that studies actual relationships between defect reports and changes of source code in Open Source software projects. However, defects and the defects management have been studied from other viewpoints. Also change data has been studied from other viewpoints. In many studies, information that was stored in defects and changes of source code was called as trails. These trails are usually defined as information left behind by the contributors to the development process, usually in the forms of logs.

The resolutions of defects have been studied by Koponen and Hotti [10]. The study showed that the defect management system, such as Bugzilla, provides a great opportunity to analyze defects in Open Source projects. However, it also showed that

the defect reporting seems to be inefficient because the majority of defect reports will never lead to changes. Furthermore, it showed that defect reporters does not browse already reported defects before committing a new one. So, the proportion of duplicate defects and invalid defect reports can be up to 60 per cent.

Studies have shown that the logs of the version management systems are also rich sources when analyzing the evolution of software. For example, German et al. [6, 7] have developed a tool that extracts, enhances and visualizes trails. Another interesting study was done by Zimmermann and Weißgerber [13]. They presented a pre-processing model for CVS data for fine-grained analysis. According to their study model will improve the quality of analysis and results.

Besides that these systems have been studied separately, there have been several studies that combines information from the defect management and the version management system. For example, Antoniol et al. [1] have also described a framework to consistently merge information extracted from the source code, the defect reports and the change history. Other studies that have combined defects and change data have used combination to release history population [4], feature tracking and hidden dependencies analysis [3, 5].

One closely related work to this study is a research that was done by Sliwerski, Zimmermann and Zeller [12]. They analyzed change data archives and defect archives to find fix-inducing changes. The fix-inducing changes lead to defect reports. These defect reports were fixed with new changes. Their research report described also a method to extract data from version and defect archives and how to link defect reports to changes. Our study uses similar approach to the data retrieval.

Although analysis of software trails has become more known after the analysis of Open Source, trail analysis has been done also for proprietary software. For example, Mockus and Votta [11] have used data of change requests of a multimillion-line telecom software system.

## 6 Conclusions

Our purpose in this study was to create an approach and software that could evaluate the maintenance process through the defect management and the version management system. Furthermore, our goal was to evaluate a relation between defect reports and changes of source code.

To study this relationship, we built software that retrieves defects, source code and their change

history from defect management and version management systems. Our system was build to support CVS and SVN version management systems, and Bugzilla defect management system. We analyzed all changes to find syntactically or semantically a link to defect reports. To evaluate our approach and software we analyzed two case studies Apache HTTP Server and Mozilla Firefox. We evaluated the proportion of source code changes that are initiated by defect reports in the case studies.

We found that both cases had a lot of changes that were not initiated by defects. The proportion of the defect initiated changes was 10 per cent in Apache. However, all defects that led to the resolution fixed had a corresponding change in the version management system. Controversially, in the case of Mozilla, 61 per cent of changes were defect initiated. Interestingly, there were plenty of defects that were fixed but there were no changes related to the defects. Furthermore, we found that less active developers were more likely to make defect initiated changes in both cases.

Furthermore, if we compare this to the ideal model, where all changes were defect initiated, results are remarkable. It shows that the communication about problems between users and developers is not fully based on the defect management system.

## 7 References

- [1] Giuliano Antoniol, Massiliano Di Penta, Harald Hall, and Martin Pinzger. Towards the Integration of CVS Repositories, Bug Reporting and Source Code Meta-Models. In the 2nd Workshop on Software Evolution through Transformations: Modelbased vs. Implementation-level Solutions, Rome, Italy, 2004. Electronic Notes in Theoretical Computer Science, Elsevier
- [2] Bugzilla.org, 2005. Bugzilla <http://www.bugzilla.org>
- [3] Michael Fischer, Martin Pinzger, and Harald Gall. Analysing and Relating Bug Report Data for Feature Tracking. In Proceedings of the 10th Working Conference on Reverse Engineering (WCRE'03), 2003 IEEE
- [4] Michael Fischer, Martin Pinzger, and Harald Gall. Populating a Release History Database from Version Control and Bug Tracking Systems. In Proceedings of the International Conference on Software Maintenance, pages 23-32, Amsterdam, The Netherlands, September 2003
- [5] Beat Fluri, Harald C. Gall, and Martin Pinzger. Fine-Grained Analysis of Change Couplings. Fifth IEEE International

- Workshop on Source Code Analysis and Manipulation (SCAM'05), 2005
- [6] Daniel M. German. Mining CVS repositories, the softChange experience. In First International Workshop in Mining Software Repositories, 2004. to appear
  - [7] Daniel M. German, Abram Hindle, and Norman Jordan. Visualizing the evolution of software using softChange. In Software Engineering Knowledge Engineering (SEKE'04), June 2004.
  - [8] Timo Koponen and Virpi Hotti. Evaluation framework for open source software. In Proceedings of The 2004 International MultiConference in Computer Science and Computer Engineering, Las Vegas, NV, USA, 2004. CSREA Press.
  - [9] Timo Koponen and Virpi Hotti. Open source software maintenance process framework. In 5-WOSSE: Proceedings of the fifth workshop on Open source software engineering, pages 1-5, New York, NY, USA, 2005. ACM Press.
  - [10] Timo Koponen and Virpi Hotti. Defects in open source software maintenance - two case studies - apache and mozilla. In Proceedings of The 2005 International MultiConference in Computer Science and Computer Engineering, Las Vegas, NV, USA, 2005. CSREA Press.
  - [11] Audris Mockus and Lawrence G. Votta. Identifying Reasons for Software Changes Using Historic Databases. In Proceedings of the International Conference on Software Maintenance (ICM 2000), pages 120-130, san Jose, California, USA, Oct. 2000. IEEE
  - [12] Jacek Sliwerski, Thomas Zimmermann, and Andreas Zeller. When do changes induce fixes? In MSR '05: Proceedings of The 2005 international workshop on Mining software repositories, pages 1-5, New York, NY, USA, 2005. ACM Press.
  - [13] Thomas Zimmermann and Peter Weißgerber. Preprocessing CVS Data for Fine-Grained Analysis. International Workshop on Mining Software Repositories, Edinburgh, 2004
  - [14] Thomas Zimmermann, Peter Weißgerber, Stephan Diehl, and Andreas Zeller. Mining Version Histories to Guide Software Changes. In Proceedings of the 26th International Conference on Software Engineering (ICSE), pages 563-572, Edingburg, Scotland, UK, May 20004. IEEE.