Evaluation Framework for Open Source Software Maintenance

Timo Koponen
Department of Computer Science
University of Kuopio
Kuopio, Finland
timo.koponen@uku.fi

Abstract— **Open Source Software is becoming evermore important and widespread these days, so its maintenance has become important issue. In this paper, we presented an evaluation framework for Open Source Software maintenance. The framework was evaluated with five well known case studies. The results showed that our framework was suitable for evaluating scale of the project and activity, efficiency, traceability and quickness of the defect management and maintenance processes. The case studies showed that quality of the defect reports was low and therefore many of them were ignored as duplicate or invalid. In addition, most of the software changes were not connected to defects, which may express lack of management.**

Keywords Problem tracking, issue tracking, bug report, software modification, defect life cycle, metrics.

I. INTRODUCTION

Open Source Software (OSS) is becoming evermore important and widespread these days. While OSS is used in several foundations, companies, and communities its maintenance has become more important issue. Definition for the software maintenance has been presented in the IEEE Standard for Software Maintenance as [4]:

> "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."

Although OSS has been delivered to users in early stage of the life cycle, it complies with the definition of software maintenance.

Our earlier studies [8] on OSS have presented a general overview of the OSS maintenance processes where these processes were managed with defect management and version management systems. The overview of processes is presented in the Section 2. Other earlier studies have also presented methods for evaluating several aspects of the OSS maintenance. Because maintenance activities are divided to correcting and improving or enhancing, their proportions can be used to determinate maturity of the software [3]. The proportion of enhancive maintenance [7] was found is low in cases, which showed that software were mature. The traceability of defect management process was evaluated through the defect life cycle [8] and study showed that maintenance activities were not traceable due insufficient usage of the defect management system. The analysis of source code changes and their origin [10] showed that most of the changes were not related to the defect reports.

While these studies presented separate methods and attributes for evaluating the OSS maintenance process, our objective is provide a framework for evaluation of the OSS maintenance process, which covers efficiency, manageability and state of the maintenance process.

Rest of the paper is organized as follows: Section 2 introduces background and the framework; Section 3 presents research method and five case studies which were used to evaluate proposed framework. Then, Section 4 presents the results of evaluated case studies; Section 5 analyses results and describe related work. Section 6 concludes the study, considers suitability of the framework.

II. BACKGROUND AND EVALUATION FRAMEWORK

OSS users report defects in the defect management system (DMS), such as Bugzilla [1]. Defect reports are representation of the defects in the DMS. Later, the developers trawl and assigns defect reports. Then they retrieve the source code from the version management system (VMS), such as Concurrent Version System (CVS) or Subversion (SVN), and make changes. Changed source code is updated to version management system as a change. Changes include changed lines of the source code, identification of authoring developer and description of the change. After submitting the changes, defect reports are marked as resolved and type of the resolution of the defects are set fixed if changes were made. If the defect did not cause any changes, resolution of the defect report is set to describe reason, such as duplicate or invalid report. The overview of maintenance process is presented in Figure 1.

DMS and VMS store every change of source code and defect report. So, they provide large amount of data that can be used to analyze OSS maintenance process. Earlier studies have presented several methods that can be used for analysis of maintenance process through project characteristics [5], type of the defects [7], defect life cycles [8], and change origin [10].

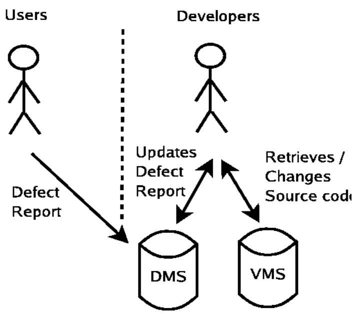


Figure 1. Overview of the process

Our framework adapts methods from these studies and combines them to a framework. The framework has eight different aspects for project and process evaluation. First two aspects describe the project and next six aspects analyses maintenance process.

The first aspect describes type and working environment of the project. Its attributes *Software type and intended audience* describes type of the software. Next, attributes *availability and type of the defect and version management systems* describe working environment of the project.

The second aspect describes scale of the project with attribute *Opened defect reports*. Larger number means larger project and more active user which report defects. In addition to defect reports, also attribute *Source code changes (SCC)* describes scale of the project and its development and maintenance activities.

The third aspect describes activity of the defect management with attribute *Resolved defects%*. If almost all defect reports have been resolved, defect management is active.

The fourth aspect describes efficiency of defect reporting. Attribute *Fixed defects%* represents proportion of resolved defects that led to changes. If almost all resolved defect reports have caused fixed, efficiency (and quality) of the defect reports is good.

The fifth aspect describes stage of the software life cycle. Attribute *Enhancements%* represents proportion of the fixed defects that were enhancements. It describes software's stage of the life cycle. Majority of the enhancements are done in the early stages of OSS development, so if the proportion of enhancements is low, software is mature.

The sixth aspect describes quickness of the defect management and maintenance with attributes *Resolving time for Non-Fixed Defects (NFD)* and *Resolving time for Fixed defects (FD)*. Attribute Resolving *time for NFD* time that was required to resolve defects that did not cause changes. Attribute *Resolving time for FD* presents time that was required to resolve defects that caused changes to the software. When resolving time is low, quickness and efficiency of the defect management is better.

The seventh aspect describes traceability of the maintenance with attributes *Defect life cycle* for RD and FD. Maintenance activities are traceable if they are documented appropriately. Traceable defect life cycle covers all major activities that are verification, analysis, implementation and quality assurance. According to our previous research [6, 8] defect life cycle should have four states that describe maintenance activities. These states are presented in the Table 1 with started activities. Started activity expresses activity of the maintenance process that starts when defect enters to that state.

TABLE I. STATUSES AND ACTIVITIES OF THE LIFE CYCLE

State	Started Activity
Unconfirmed	Verification
New	Analysis
Assigned	Implementation
Resolved	Quality assurance

A typical life cycle for defect report should be *Unconfirmed->New->Assigned->Resolved* defect causes changes and become fixed; and its maintenance activities are traceable. So, shorter life cycle expresses that maintenance activities are not traceable for those defect reports. However, if defect does not lead changes because it is duplicate or invalid, defect may not be verified so it can be resolved directly. Therefore, life cycle can be *Unconfirmed->Resolved* for a defect which did not caused changes.

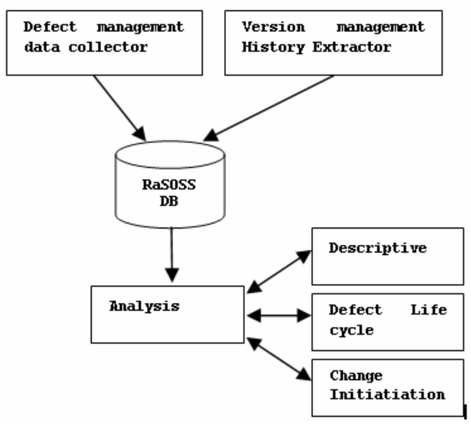
The eighth aspect describes manageability of the change management. Attribute *Defect initiated changes%* represents proportion of the source code changes which were caused by defect reports. It describes management of DMS and VMS. If proportion of defect initiated changes is high, DMS and VMS are used concurrently for managing maintenance process. Low proportion, controversially, expresses that DMS and VMS does not have significant roles in the maintenance process. Although, in the early stages of the software life cycle, defect management system may not be used, so the proportion can be low also for those reasons. Attributes of the evaluation framework is presented in the Table 2.

TABLE II. ATTRIBUTES OF THE EVALUATION FRAMEWORK FOR OPEN SOURCE SOFTWARE MAINTENANCE

Attribute	Value type
Software type	String, such as Server
Intended audience	String, such as Developer
Availability and type of Defect management system (DMS)	No/Yes, such as CVS/SVN/ Other
Availability and type of Version management system (DMS)	No/Yes, such as Bugzilla/ Tracker/ Other
Opened defects	Number
Source code changes	Number
Resolved defects (RD)	Percent
Fixed defects (FD)	Percent
Enhancements	Percent
Resolving time for non-fixed defects	Number, days
Resolving time for fixed defects	Number, days
RD life cycle	String
FD life cycle	String
Defect initiated source code changes	Percent

III. Case Studies and Data

To evaluate our framework, we selected sample cases that are widely used, well known, and they cover wide range of software types. While our framework has large set of attributes, data gathering and analysing is challenging. So, we build software, RaSOSS, which retrieves and analyses data from DMS and VMS [9]. The overview of RaSOSS is presented in the Figure 2.



Defect management data collector
Version management History Extractor
RaSOSS DB
Analysis
Descriptive
Defect Life cycle
Change Initiation

Figure 2. Overview of the RaSOSS

As the figure shows, there is three main modules in the system, which are defect data collector, version history extractor, and analysis module. The defect data collector retrieves defect data from the defect management system. The version history extractor retrieves the source code of the software and the change history of source code files from the version management system. After the data retrieval both modules export data to database where the analysis module could classify and analyze defects and changes. The RaSOSS produces attributes which were described above in the framework. Descriptive analysis module produces most of the attributes expect *defect life cycles*, which are produced by defect life cycle module, and *defect initiated source code changes*, which is produced by change initiation module.

The case studies were Apache HTTP server, Mozilla Firefox, The GIMP, Glib and KWord.

Apache HTTP Server is currently the most popular web server in the Net. It is developed and maintained nowadays by the Apache Foundation [14]. *Mozilla Firefox* is Open Source Web browser that is produced by the Mozilla Foundation. Mozilla project is based on Netscape's Communicator suite which source code was released in 1998 [15]. *The GNU Image Manipulation Program (Gimp)* is Open Source application for creating and manipulating graphic images. The Gimp has been created by Kimball, Mattis, and hundreds of other contributors [16]. *Glib* is low-level core library that forms basis of the GNOME. It provides basic data structure handling, portability wrappers and interfaces for basic run time functionality [17]. *KWord* is word-processing application that is part of the KOffice Project. The purpose of KOffice project is to create a free office suite for K Desktop Environment (KDE) [18].

IV. Results

We gathered and analyzed defect reports and source code changes from selected two-year time-period between September 2003 and September 2005. The Table 3 presents attributes that were produced from the analyzed data. Defect life cycles are presented separately in the Table 4.

The number of opened defects shows that defect reporting was very active in the Mozilla Firefox and the other cases were less active. The difference between Mozilla Firefox and other projects is caused by user amount and intended audience groups.

The proportion of resolved defects expresses that maintenance was quite active in all projects because over two thirds of the opened defects were resolved. However, the quality of defect reports was very low in the Mozilla Firefox because a bit more than one tenth of the resolved defects become fixed and caused changes to software. Most of the defect reports that did not caused changes were duplicate or invalid. In the other projects, quality was better.

The proportion of enhancements was about ten percent in all case studies except Gimp. Because OSS is released in early stage of life cycle, high proportion expresses, controversially, that software is under development - not mature. However, overall proportion was so low that all of them were mature.

However, in the Apache, Gimp and Glib, non-fixed defects were resolved quickly in few days; In the Gimp and Glib also fixed defects were solved in few days. In the other cases, non-

TABLE III. STATISTICS OF THE CASE STUDIES

Attribute	Apache	Firefox	Gimp	Glib	KWord
Software type	Server software	Desktop software	Desktop software	System library	Office software
Intented audience	Admin/ Operator	End-user	End-user	Developers and programmers	End-user
Availability and type of DMS	Yes, Bugzilla	Yes, Bugzilla	Yes, Bugzilla	Yes, Bugzilla	Yes, KDE Bugzilla
Availability and type of VMSS	Yes, SVN	Yes, CVS	Yes, CVS	Yes, CVS	Yes, CVS
Opened Defects	1266	27681	3088	786	568
Source code changes	2877	2884	7102	1262	606
Resolved defects	74 %	73 %	88 %	86 %	65 %
Fixed Defects	31 %	12 %	41 %	58 %	56 %
Enhancements	7 %	9 %	22 %	12 %	10 %
Resolving time for non-fixed defects	24 days	1 days	0 days	2 days	33 days
Resolving time for fixed defects	65 days	34 days	3 days	5 days	65 days
Defect initiated source code changes	10 %	61 %	20 %	29 %	14 %

TABLE IV. MOST COMMON DEFECT LIFE-CYCLES IN THE CASE STUDIES.

Project	Resolved defects life-cycle	Fixed defects life cycle
Apache	New →Resolved →Closed	New →Resolved →Closed
Mozilla	Unconfirmed →Resolved	New →Resolved
Gimp	Unconfirmed →Resolved	Unconfirmed →New →Resolved
Glib	Unconfirmed →Resolved	Unconfirmed →Resolved
KWord	Unconfirmed →Resolved	Unconfirmed →Resolved

fixed defects were resolved in one month and fixed defects in two months.

The relationship between defect and version management systems was very loose in all cases, except Mozilla Firefox were over sixty percent of the changes was caused by defect reports. Even the proportion of defect initiated changes was low in the other projects; it does not necessarily mean that systems are not used concurrently. It is also possible that developers might have forgotten to express relation. However, usage of defect management and version management seemed to be inefficient.

The Table 4 show that defect life cycle for resolved defects was similar to expected one in all case studies. However, life cycle for fixed defects was much simpler than expected in any of the case studies. In the Gimp, defects were verified before closing but in the other cases defects were not verified, analyzed or implemented. So, the maintenance activities were not traceable in the case studies.

V. DISCUSSION AND RELATED WORK

To compare the framework to earlier studies and their results, we first present shortly few attributes that are used with the proprietary software but were not applicable for OSS. After that, we present attributes that were adaptable for OSS.

While the traditional attributes for software maintenance, which are based on reliability, risk or test attributes, such as mean time between failures and total maintenance time [11], were not applicable for evaluation of the OSS maintenance because such data was not available. Also, several attributes have been presented that measure customer satisfaction, costs efficiency and scheduling [12] but none of them were usable in our case because there were no definable customers.

However, attributes, which measured required time for change, number of defects, type of the changes and non-fix causing defects [12], were applicable and required data was available through defect reports. Also, attributes [3], which determinate state of the product life cycle through defect reports and enhancements requests, were also applicable and data was available through defect reports and enhancement requests. We found that all evaluated case studies were in mature stage, although software was under modifications that were not related to defects.

So, attributes of the framework has been used earlier in evaluation of proprietary software maintenance, and now they are found applicable for OSS maintenance. The framework and case studies showed framework can be used if there is defect and version management system that are capable to provide necessary data. At the moment, we have found that Bugzilla [2], Concurrent Versions System [2] and Subversion [13] meet these requirements.

VI. CONCLUSIONS

In this study, we created a framework for evaluation of OSS maintenance. The framework includes several attributes for evaluating type, activity and quality of the maintenance. The framework was evaluated with five case studies and case studies. Case studies showed that framework can be used to evaluate maintenance of OSS to provide overview. However, manual evaluation was found very challenging and time consuming due large amount of data and therefore we implemented software that can be used in data gathering and evaluation.

Case studies showed that most of the projects were stabilizing and maintenance was active in all the projects because most of the defect reports were resolved. However, the quality of defect reports was not good because most of the defect reports did not cause any changes to software and defect reports caused additional work to maintainers. In addition, maintenance activities were not traceable in all case studies; and most of the changes were not related to defect reports in most of the case studies. So, maintenance does not use efficiently defect and version management systems.

REFERENCES

[1] "Bugzilla.org", 2005. Bugzilla http://www.bugzilla.org (15.05.2006).

[2] "CVS - Open Source Version Control". http://www.nongnu.org/cvs/ (12.05.2006).

[3] H-J. Kung and C. Hsu. "Software Maintenance Life Cycle Model." In Proceedings of the International Conference on Software Maintenance 1998. 1998. IEEE.

[4] IEEE Computer society. "Guide to the Software Engineering Body of Knowledge (SWEBOK)". Los Alamitos, California. ISBN 0-7695-2330-7

[5] T. Koponen and V. 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.

[6] T. Koponen and V. 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.

[7] T. Koponen and V. 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.

[8] T. Koponen. "Life-cycle of the defects in Open Source Software Projects". In Proceedings of The Second International Conference on Open Source Systems, Como, Italy, 2006. Springer IFIP.

[9] T. Koponen. "RaSOSS – Remote Analysis System for Open Source Software". In Proceedings of the International Conference on Software Engineering Advances, Tahiti, French Polynesia, 2006. IEEE.

[10] T. Koponen and H. Lintula. "Are the Changes Induced by Defect Reports in the Open Source Software Maintenance?" In Proceedings of The 2006 International MultiConference in Computer Science and Computer Engineering, Las Vegas, NV, USA, 2006. CSREA Press.

[11] N. Schneidewind. "Measuring and Evaluating Maintenance process Using Reliability, Risk and Test Metrics". In Proceedings of the International Conference on Software Maintenance 1997. 1997. IEEE.

[12] G. Stark. "Measurements for Managing Software Maintenance". In Proceedings of The International Conference of Software Maintenance 1996. 1996. IEEE

[13] "Subversion.tigris.org". http://subversion.tigris.org (12.05.2006).

[14] "The Apache HTTP Server Project". http://httpd.apache.org (12.05.2006)

[15] "Home of the Mozilla Project". http://www.mozilla.org (12.05.2006)

[16] "The GNU Image Manipulation Program". http://www.gimp.org (12.05.2006)

[17] "The GIMP Toolkit". http://www.gtk.org (12.05.2006)

[18] "The KOffice Project - KWord". http://www.gtk.org (12.05.2006)