Aspects of Software Quality Assurance in Open Source Software Projects: Two Case Studies from Apache Project*

Dindin Wahyudin, Alexander Schatten, Dietmar Winkler, Stefan Biffl
Institute for Software Engineering and Interactive Systems,
Vienna University of Technology, Vienna, Austria
{dindin, schatten, winkler, biffl}@ifs.tuwien.ac.at

Abstract

Open source software (OSS) solutions provide mission-critical services to industry and government organizations. However, empirical studies on OSS development practices raise concerns on risky practices such as unclear requirement elicitation, ad hoc development process, little attention to quality assurance (QA) and documentation, and poor project management. Event then the ability to produce high quality products in such an environment may seem surprising and thus warrants an investigation on effective QA mechanism in OSS projects. This paper provides a preliminary exploration to improve our understanding of software quality practices in different types of OSS projects. We propose a framework of QA in an OSS project, elicit OSS stakeholder value propositions for QA, and derive performance indicators. For an initial empirical evaluation we applied these indicators to 5 releases of 2 large Apache projects (Tomcat and MyFaces) to analyze the extent to which QA aspects are commonly performed during development process.

***Keywords**: Open Source Software and Software Quality, Distributed Software Development, Value Based Software Engineering, Product Evolution*

1. Introduction

Industry and government organizations such as Wall Mart and General Motors increasingly adopt open source software (OSS) as alternative solution to proprietary commercial software products. Some well known OSS products, e.g., HTTP Server or Apache Tomcat, can be considered successful as they currently dominate their particular application domains as reported by Netcraft[1] and The Serverside[2].

OSS projects can be viewed as extreme form of distributed software development based on criteria suggested by [4], such as: (a) highly distributed and volunteer contributors, (b) considerable cultural and time zone differences of contributor teams, (c) weak formal design, and (d) weak formal project and quality management. As software quality is believed to be strongly dependent on the quality of the software development process, quality assurance (QA) professionals and customers are particularly concerned about the quality of released OSS products and are interested in risk assessment of the associated OSS development processes. This background raises the need for investigating QA activities commonly performed by more or less successful OSS project communities in their development processes. From QA point of view, we can view Linus Torvalds's [8] style of OSS development to produce high quality software, which encompasses the following practices: release early and often, delegate everything you can, be open to the point of promiscuity as a spawning ground for collaboration among developers and users to perform bug detection, bug assessment, bug tracking, and code peer review. In a commercial software project, in more semi-formal way some of these activities seem similar to inspection [2], which is an effective but also expensive approach to QA; however the community in OSS project has tackled the financial barrier as they build on self motivated volunteers.

This paper seeks to make preliminary exploration of aspects of QA in an open source software project. We argue that such exploration is important for the OSS project stakeholders in order to achieve better quality OSS products, and as basis for future work in process improvement of similar development style. In this paper we proposed two research contributions which are: (1) based on expert interview and literature research we know that there are several common

*We would like to thank Martin Marinschek the co-founder of Apache MyFaces, for his suggestions in developing this paper. This paper has been partly supported by The Technology-Grant-South-East-Asia No. 1242/BAMO/2005 Financed by ASIA-Uninet

[1]http://news.netcraft.com/archives/web_server_survey.html (20/02/2007)

[2]http://www.theserverside.com/ (10/01/2007)

QA practices in a typical OSS project and how the project community performs such activities based on their value expectations (2) we developed some performance indicators for OSS-QA processes, which can be applied in different types of OSS project.

2. Related Work

This section briefly summarizes related work on the OSS development process, quality aspects during project execution and the OSS project community structure.

2.1 The Many Types of Open Source Software Products

We can define the type of an OSS project based on application domain, project size and the involvement of salaried developer during development process.

2.1.1 OSS Application Domains

Our observations on projects listed in Sourceforge[3] reveals that from 18 categories, the top 5 are Internet application (15.4%), Software development (15.1%), System (12.4%), Communication (10%), and Game/Entertainment (9.3%). However based on project maturity ranks in Sourceforge depicts that most of the projects (more than 70%) are still in early stages or already at the end of their lifecycle, and only a small portion of the projects have reached their maturity and produce stable releases.

2.1.2 OSS Project Size

Other studies from 100 mature projects listed in Sourceforge by [6] sugests that the developer community may consist of a single fighter up to more than 250 developers at one time. The study categorized projects size into 6 classes of number of developer per project. The results disclosed that 86.2% of the projects employ less than 6 developers, and less than 1% of the projects have more than 16 developers involved during the development processes which we defined as large project.

2.1.3 OSS Project Sponshorship

In pure open source projects, all the workers are volunteers, with rare formal processes that are followed or formal specifications produced. It is worth noting, however, that currently a number of important OSS projects are hybrid project which supported by companies and some participants are not volunteers (e.g., JBoss, Apache JackRabbit , Myfaces, Sourcefire or OpenOffice), such projects are likely to fit in some of their sponsor traditional tools and in-house experts to the open source community.

[3]http://www.sourceforge.net(10/03/2007)

2.2 Software Quality Assurance in OSS Projects

Empirical studies in [1] suggest the open source software model has led to the creation of significant pieces of software, and many of these applications show levels of quality comparable to closed source software development. Raymond suggests the high quality of OSS can be achieved due to high degree of peer review and user involvement in bug/defect detection. From QA point of view these activities are quality assurance practices which refer to a semi-structured process of trying to find and correct bug/defects during various phases of the software development process. In large OSS projects such as Gnome, Mozilla, and Debian, QA has become a part of the community awareness. These projects explicitly stated their (a) QA goals such as to improve the product where it is needed, and to keep the quality of the distribution as high as it should be (b) who the QA participants are, (c) what typical QA actions are, guidelines, and to which part of software QA gets applied. Based on these particular OSS projects reports we can assume that to some extent, these products satisfy particular needs and quality standards of users.

2.3 The Liveness of Open Source Software Development

The social structure in an OSS project is typically composed of 2 major groups: the developer community and the user community. In order to survive, the main objective of an OSS project is to build up and keep loyal and healthy project communities and produce software that continues to satisfy particular user needs. A healthy project is signified by the liveness of the community such as rapid development activities, high use intensity and receives useful feedback for further improvement [10]. From the developer's point of view, an OSS project is in a state of continuous release. In OSS project a developer is not merely coder or other developing contributor, but they often also act as the end user of the product. Hence, they usually run the latest available code to spot defects and stay away from unstable features [5].

A recent study [9] suggested that the developers social structure in OSS projects can provide some hierarchy of management and controlling based on self organizing patterns, which is also called as "meritocracy" in Apache Software Foundation (ASF). The meritocracy in OSS suggests that the contributor (which are the larger part of the development community) start their roles in bug handling activities, which depict his merit to the community. When the project leading teams felt that particular developer has "earned" the merit to be part of the development community, they granted committership (direct access to the code repository). Mockus et al. [7] suggested such structure to increase the development effectiveness and maintainability. An OSS project is typically led by a group of key committers or steering board called "Project Management Committee" (PMC) in Apache projects. One of the main roles of the PMC is to ensure that balanced and wide scale peer review and collaboration do happen, e.g., how to review a code contribution, who should review the code contribution and when to stop a peer review. The second group is the user community, who uses the OSS products for particular reasons. In OSS most of the developers of the OSS project are also users. Hence compared to commercial software products, the user community in OSS project is expected to be more active to provide feedback for functionalities of every product release.

3. Research Methodology

Based on our research questions presented in introduction, hence in this study, we conducted intensive literature research, and 2 OSS expert interviews. Both activities aimed to answer the initial research questions "What are the QA practices in a typical OSS project?"; "How does the project community perform such activities?". The interviews were of a fairly informal nature to allow a thorough exploration following the expertise of interview partners on quality in the projects the interviewees were involved in. The literature review was to summarize current research understanding of QA aspects in OSS project (see related work). The results were used to define critical stakeholders of QA in an OSS project, their win conditions, and related QA performance measurement. Building on these results we proposed a framework of QA aspects in OSS project, to illustrate the flow of QA activities and stakeholder interaction during the development process. The framework development started from bug reporting and bug fixing scenarios proposed in [5], and Bugzilla 3.0rc1 documentation[4], later base on expert interviews we extent the model to a complete defect life cycle by introducing three process groups which are: defect detection, defect verification and solution verification.

As the second contribution, we derive several performance indicators of QA processes based on stakeholders' values; here we adopt some indicators that commonly used in commercial project such as service delay and effectiveness of defect collection into OSS project context. We applied these indicators on 2 Apache projects (Apache Tomcat and Apache MyFaces). These projects are considered as large projects as they employ more than 20 committers and more than 50 developers and already have more than one major release when the study was conducted.

[4]http://www.bugzilla.org/docs/ (10/03/2007)

4. Framework for Aspects of Software Quality Assurance in OSS Project

This section describes the stakeholder value expectation from quality assurance in OSS project, their win conditions and related performance measurement for QA. Later we proposed a framework of QA aspects in OSS project as an extension of stakeholder roles and activities.

4.1 Stakeholder Value for Quality Assurance in OSS Project

The stakeholders in OSS project are represented by each individual in the community. To better understand their expectation about product quality in OSS project we need to elicit their values, start by eliciting win conditions based on their roles in the project and define performance measurements [3]. We interviewed experts to find out the stakeholder' needs for good quality OSS product, we conduct the interview in two session first is by direct interviewing the experts with some open questions and second by asking more QA focused questions through email. In this research we focus on bug lifecycle as the prominent part of software process improvement in OSS project.

User. The win condition of users (or developers) who use the software, detect and report a defect is to gain faster response from the development community, faster defect closure, and stable solutions. The performance measurements are defect response time, defect closure time and defect reporting frequency.

Developer. Developers are the main stakeholder in QA activities, as they involve in most aspects of defect handling (reporting, fixing, and reviewing). The larger group of developers working on the head and incorporating the current version in their productive applications, and when they do, they really find out about problems in the OSS product as well. The win conditions of developer are to have proper access to current development repositories and collaboration tools to support their works, merit based incentive from the community, less invalid defect report, adequate information of reported defect, and flexible time to resolve defect. Examples of performance measurement are defect collection effectiveness, and defect correction effort.

Committer. Committers have responsibility for assuring the quality of the software product before release. One typical task is to review any defect resolutions, and to decide whether a patch should be added to release log. A committer may expect to have adequate information of defect, the contribution poses no threat to current body of code, and higher number of verified resolution which means the defect were solve properly according to their specifications. An example of performance measurement from committer point of view is the proportion of verified solution per total number of defect solution

Project Manager. A project manager as described in related work needs to monitor these performance metrics to ensure appropriate quality assurance activities are well performed. For better illustrating the QA aspects and interaction among project community during development process, we proposed a framework of QA aspect in OSS project, which described in following section.

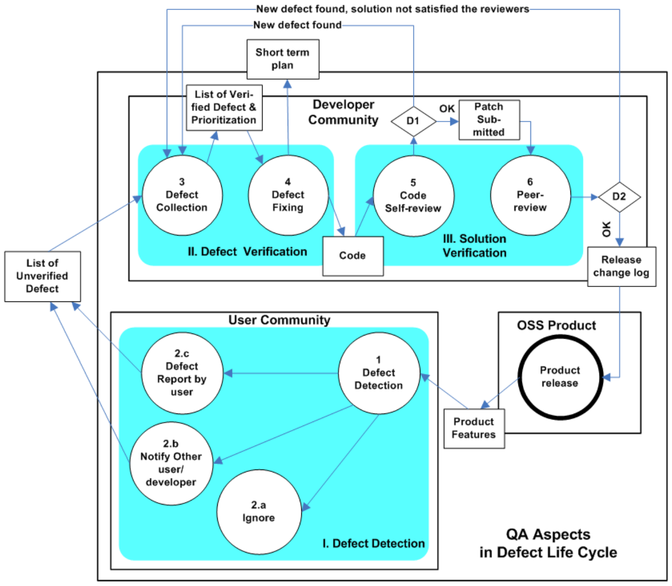

Figure 1. Framework for quality assurance in an OSS project. The quality of OSS is the result of interrelated QA activities within the developer and user communities.

4.2 Processes in The Framework

This subsection described the detail of our proposed framework of QA aspect in OSS project based on typical processes performed by different roles of project participant during bug or defect life cycle. In this work we define defect as an error, flaw, mistake, failure, or fault in software that prevents it from behaving as intended. A defect reported by user (or developer) at user community; however its existence should be proven and validated as defect before further processed by the developer community as illustrated in Figure .

4.2.1 Process Group I: Defect Detection

The detection and reporting process, provide information of a defect existence, and sometime accompanies with early assessment of a defect, accompanying with early defect analysis.

Process 1: Defect detection, in open source is more like black box testing [5], as the user (common user or developer) have been using particular features of software release and spotted a defect, error or failure.

Process 2(a,b,c): reporting a defect into the project bug tracker heavily relies on the motivation of the user, as he may just (2a) ignore the defect and continue using the software, or (2c) he fills the issue into the tracker, accompanied with defect summary, defect description and other information needed by the tracker. However as most of the common users are partially unknown to the project hence most common practices in defect reporting are to (2b) notify other users or developers about the finding and ask for their opinion in the mailing list or forum. A more experience developer who noticed the issue then perform some early analysis, locates the defect and roughly estimates its effect(severity) if the defect is valid. Then he may fill a report in, otherwise he notifies the community to ignore as it is a false defect.

4.2.2 Process Group II: Defect Verification

The defect verification consists of defect collection and defect fixing/correction. Both processes are very important and similar to the same named processes in commercial software inspection. The objective of group II is to validate the existence of a defect as defect of particular software release, and later perform necessary actions to correct the defect.

Process 3: Defect Collection. The defect collection begins after a defect report listed into the tracker. The process is similar to white box testing, here defects are first stated as ”open”, since it is unverified and there is no action has been taken yet. One or more developer may read the report, add some comments and ask for more information from the reporter. Later in order to validate defect existence, he needs to reproduce the defect, and then analyze defect location and its effect (severity) to the software product. Once a defect successfully reproduced and analyzed, the developer may confirm the defect existence (defect stated as ”new”), the defect already reported by others (”duplicate”) or false defect (”invalid”) which should be ignored. The expected result from sub process 3 is to have a list of verified defect with enriched description and specification.

Process 4: Defect Fixing is a set of activities to correct a defect. A developer who has interest in a verified defect may take ”ownership” of the defect, create a short term plan which announce that he is working on a particular defect and set of code and expecting other developers to avoid them or attempt to synchronize their changes. If succeed, the process may deliver a set of code which should be first self reviewed later to be submitted to the mailing list, or tracker further review by other developers or committers.

4.2.3 Process Group III: Solution Verification

Process 5: Code self review. During defect solution development an assigned developer produces a set of code locally and submits the results. It is worth noting that in OSS project it is very unlikely for a patch to be submitted without first being self reviewed. Eventually after self review the developer may decide (see decision: D1) to submit the results as patch into the developer community (e.g through tracker and mailing list) or it is possible that the development of a solution will impose some new defects in the code, which should be reported as a feedback input for process 3.

Process 6: Peer review. Almost every code contribution, patch or commit is cross checked by attentive people (e.g. other developers or committers) in the developer community. Later reviewed patch may be added to the body of code or written as change into release change log by the committer, however if the patches could not satisfied the majority reviewers for several reasons such as the patch does not meet the defect specification or some new defects were found, in these cases most likely the defect will be returned to the bug tracker and stated as ”re-open” or ”new”.

5. Case Study Proceeding

The case study was completed in March 2007, performing several steps: case study goals definition, defining study variables, and derived hypotheses.

5.1 Goals Definition

From project manager point of view, QA is very important in order to produce high quality software which satisfy user needs, and make sure current QA performance meet stakeholder win condition as depicted in section 4.1. One focus of our research is shaping performance indicators to observe QA aspect status in a pure (Tomcat) and hybrid (MyFaces) OSS project. Therefore the part of analysis has two goals: (a) it adds further evidence to differences QA processes performance between diverse types of projects, (b) it empirically evaluates the QA performance indicators that is directly applicable in every OSS project without specific expert know-how.

5.2 Variables

The types of variables defined for the experiment are independent and dependent variables. The independent variable is the type of project (either pure voluntarily or hybrid). The dependent variables capture the QA performance in different project type. Following standard practice in empirical studies we focus on time variables and performance measures. For time variables we focus on (a) defect closure time which we defined as time spent on defect stated as ”open” until the same defect stated as ”resolved” in the tracker. This indicator represent how well the developer community in responding and provide solution for each reported defect of particular software product.

Regarding to performance measures we focus on (b) defect detection frequency defined as how many defect reported by how many reporter over the time into the tracker, this indicator shows how active the user community in reporting defect, which can reflect the usage of particular release (c) defect collection effectiveness, we defined as: ratio between number of valid defect after reviewed by some developers per number of defect reported by users in certain time, this indicator tells us the how many false alarm that reported by the user community which in some level can be annoying from the developer point of view. (d) Proportion of verified solution, defined as ratio between defects resolved with resolution closed per number of defect resolved with resolution fixed, this indicator signify the willingness of the community to resolve defect properly (e.g. by peer review every solution) before a release. In a while to better understand the QA emphasize on defect severity, we classified defect into three class of severity based on Bugzilla documentation which are Class 1 is the highest priority which related to security (critical) and fault (blocker); Class 2 those which related to feature (major, minor, enhancement, normal) and; Class 3 are those related to cosmetics work (regression and trivial). Defect severity is typically set by the developer who reviews the defect into the tracker, hence we use defect severity to draw the red line between developer value expectations with evaluated QA performance indicators.

5.3 Hypothesis

In the case study we will evaluate the following research hypotheses:

Defect Detection Frequency. A defect detection frequency signifies average number of defect report filled into the bug tracker by group of reporter in certain time. We expect in much larger and heterogeneous community such as Tomcat, has higher number of defect detection activities than MyFaces. Hence we replicated a negative hypothesis as: *H1: Defect detection frequency (Tomcat)*$\leq$ *Defect detection frequency (MyFaces)*

Defect Collection Effectiveness is probability of valid defects against overall reported defects into the tracker during case study period. We expect higher defect collection effectiveness in MyFaces, as a hybrid project should have more documentation to prevent invalid defect report and the reporter may have deeper knowledge of the project compare to pure OSS project. Hence our replicated hypothesis is *H2: Defect collection effectiveness (Tomcat) $\geq$ Defect collection effectiveness (MyFaces)*

Defect Closure Time is similar to service delay in commercial project. We expect hybrid projects should perform slower closure time of defect solution compare to pure project due their rigid documentation, guidelines and defect resolution policies. We proposed a negative hypothesis *H3: Defect closure time (Tomcat) $\leq$ Defect closure time (MyFaces).*

Proportion of Verified Solution. Verification after a defect has a positive resolution (e.g. patches) is important to make sure the quality of the solution meet the specification and not endangered current body of code of the software. We expect in less formal project environment such as Tomcat, a defect resolution will likely to be resolved faster but less frequent to be peer reviewed (defect stated as ”closed”) compare to MyFaces. Thus we proposed following negative hypothesis: *H4: Proportion of verified solution (Tomcat) $\geq$ Proportion of verified solution (MyFaces).*

5.4 Objects

The case study objects is five releases of two Apache Project Tomcat and MyFaces. The first project Apache Tomcat is a network server (system) application with a very large and diverse community background. Tomcat is pure volunteer work with 4 major releases; in this work we investigate Tomcat 5 and Tomcat 6 as the older releases (version 3 and 4) have already been abandoned by the community. The second project is Apache MyFaces, an application considered as web framework (Internet application), the project employs more homogenous participants compared to Tomcat. Apache MyFaces consists of 4 subprojects; 3 of them (Tobago, Trinidad, and Tomahawk) are extended components that offer more functionality and flexibility than using standard Core components. Project Trinidad is a donation from Oracle to ASF, while Tobago is a hybrid project as some developers are paid and closely supported by commercial organizations. In this work we examined both projects during their last 5 months of development (1/10/2006 to 1/02/2007). We retrieved SVN logs from each project defect database, and examined more than total of 500 reported defects. We calculated the proposed performance indicator based on retrieved data. Descriptive statistic is used to investigate all performance indicators.

6. Empirical Result

In this section we present empirical result and evaluate the research questions. The comparison of results from different reviewed projects is also included in following discussion.

6.1 Defect Detection Frequency

Table 1(a) displays average and standard deviation of monthly effort in 6 reviewed major releases. Tomcat 5 has the highest average number of defect report and reporter, which signify the project has larger reporter community. The ratio of each project exhibits that most of the time there are more than one defect report filled by a single reporter. The table shows that based on mean of reported defects and number of reporter, Tomcat 5 outsized all other project releases, which means the project has more active and heterogeneous reporter community.

6.2 Defect Collection Effectiveness

All projects in this study show low level of report invalidity, as the majority of reported defects had been validated and listed as positive defect instead of false ones as depicted in table 1(b). As we expected in five months of observation all MyFaces releases has less invalid defect (closer to ”1”) which illustrated more effective defect collection compare to Tomcat 5 and Tomcat 6.

Table 1. Defect Detection and Collection Activity

(a) Monthly Defect Detection Frequency in Number

		Major Releases					
	Bug Severity	Tobago	Trinidad	Tomahawk	Core	Tomcat5	Tomcat6
Mean	reported bugs	15.20	20.80	23.80	13.00	31.80	8.20
	active reporter	7.40	13.00	19.60	11.60	27.40	6.20
Stdev	reported bugs	4.82	5.12	5.26	5.43	7.12	4.09
	active reporter	1.82	2.45	4.98	5.13	6.95	2.28

(b) Defect Collection Effectiveness

	Major Releases					
Bug Severity	Tobago	Trinidad	Tomahawk	Core	Tomcat5	Tomcat6
Class 1	1	0.75	0.9	0.63	0.2	0.33
Class 2	0.94	0.94	0.95	0.92	0.63	0.87
Class 3	0.67	1	1	1	0.82	1

6.3 Defect Closure Time

Table 2 illustrates the average of defect closure time (in days) for each major release, where ”0” means a defect is resolved within the same day after the report filled in the

Table 2. Bug Closure Time Per Class of Severity in Days

		Major Releases					
	Bug Severity	Tobago	Trinidad	Tomahawk	Core	Tomcat5	Tomcat6
Mean	Class 1	62	34	11.17	43	4	0
	Class 2	45	13	9	39	22.37	4.5
	Class 3	33	2	29	N/A	13	N/A
Stdev	Class 1	24.79	40.8	16.44	57.38	7.98	0
	Class 2	27.39	26.82	76	47.48	11	9
	Class 3	39.47	0	32.99	N/A	28.72	N/A

tracker and "N/A" means there is no resolved defect in certain severity class. In table 2 in average all MyFaces subprojects need more time to solve class 1 defects compare to Tomcat releases. However the standard deviations in Tomcat releases show more diverge time to closure a defect than in MyFaces, hence it will be more complicated for project manager in Tomcat to decide when a defect is delayed. Furthermore to better understand the defect closure time, instead of categorized defect into severity classes, we distributed defects into several class of closure time as illustrated in Figure 2. In this figure MyFaces Tomahawk, and both Tomcat releases show higher performance in resolving defect, as the more than 50% of validated defects were fixed within one day. In contrary 66% defects listed in Tobago were fixed in more than 30 days which signify slower service time.

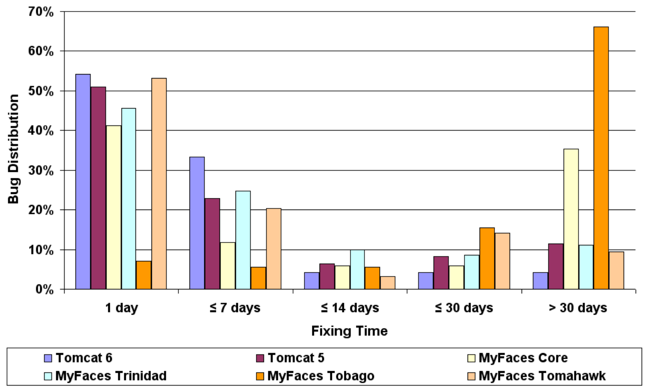

Figure 2. Bug closure time distribution in reviewed projects

6.4 Proportion of Verified Solutions

Figure 3 show that in all projects some of reported defect have been fixed with particular resolution. Compare to both Tomcat releases, all four My Faces subprojects signify higher QA activities from assigned developer as more than 50% of resolved defect have been self reviewed and tested (defect resolved as "fixed"). Further more higher effectiveness of peer review illustrated by My Faces Core, and Tobago, as in both project, most of "fixed" defect ($\geq$ 50%) were also stated as "Closed" means the defect has passed several QA processes by attentive developers or committers in the developer community.

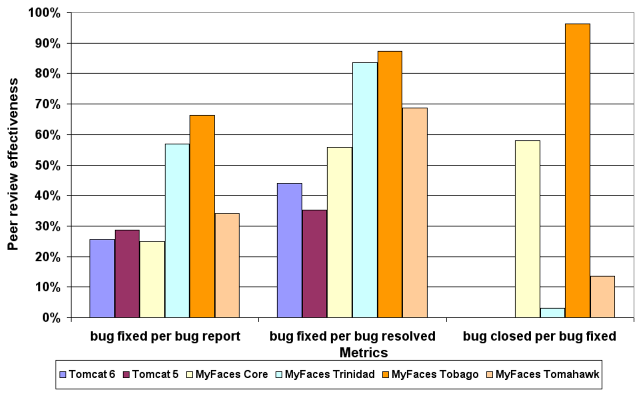

Figure 3. Verified solution proportion for resolved bugs in reviewed projects

7. Discussion

In this section we summarize the empirical result from our case study concerning QA aspects in two Apache projects. Analyzing our empirical results, we derive following implications for performance measurement of QA processes in similar OSS projects.

Defect detection frequency. The result shows that in hybrid project community such as MyFaces obtain less number of defect reports over the time submitted by only particular people in the community compare to large and heterogeneous project such as Tomcat. In summary the data signify disagreement with our hypothesis H1. It is worth noting that in Tomcat, we also found that more project participants involved in Tomcat 5 defect collection activities rather than in Tomcat 6. The reason is in particular critical application such as web server, instead of using the latest release (Tomcat 6), more users are still using the previous version (Tomcat 5) for several considerations such as security, set-up overhead, etc.

Defect collection effectiveness, the results exhibit higher probability of invalid defect reports in Tomcat releases compare to MyFaces subprojects especially in defect class 1 and class 2. For these classes of defects we can reject hypothesis H2: Defect collection effectiveness (Tomcat) $\geq$ Defect collection effectiveness (MyFaces). Here we can expect in more formal/structured hybrid project such as MyFaces, the community is has more knowledge about the software releases, thus most of the time the defect reports are valid and should be taken into consideration by the developer community.

Defect closure time. In this work we define defect closure time as time to resolve a reported defect. The results show that majority of projects resolved defect in less than 30 days (See Figure 2) and signify a responsive developer community. In case of MyFaces Tobago exhibits more fixing time were needed, which rejects our hypothesis (H3). One possible reason is that most of the defect were also required to be properly peer reviewed by other developers (See Figure 3) which eventually took more time before a defect stated as ”closed” or ”verified”. Most of these delayed defects are in middle to lower severity classes (class 2 and class 3), which have less significant impact and tends to be delayed by the developers.

Proportion of Verified Solution. Code review at the end of defect life cycle consist of self reviewed and team review. Due to limitation of investigation period, although in most of the projects we found practices of code self review (defect stated as ”fixed”), however in both Tomcat releases we barely found any evidences of code team review, as all of fixed defects are only stated as ”resolved” instead of ”closed” or ”verified”. The result also enclosed in MyFaces Tobago, the ratio of closed defect per fixed defect is very high (96%), means most of the fixed defects had been peer reviewed. Therefore we have to reject our replicated hypothesis H4. Hence we assume the hybrid community is more responsive to each patch/code submission and highly aware about its quality. We asked expert in Apache foundation, it is probably due to complexity, maturity of its releases, and releases policy that the Tomcat's developers need to spend more time to review and verify a code or patch contribution which could not be captured within our case study time limitation. As the tradeoff of slower solution verification, Tomcat offers less ”re-open” defects (stable solutions) compare to smaller OSS projects.

8. Conclusions

Quality assurance (QA) methods such as software testing and peer review are very important to reduce the adverse effects of defects in software engineering. In this paper we explore current practices of QA and possibilities for their extension in open source software (OSS) projects. This paper presented a framework for QA aspects in OSS project based on our observation from typical OSS projects. Beyond the framework we performed case study on 2 large Apache projects Tomcat and MyFaces. Our main results were: (1) based on expert interviews and literature review we found different value expectations from the members of development community for performing QA activities in an OSS project, as described in section 4.1, based on their win condition we can derived some performance measurement of QA processes which important to be monitored by the project leading teams, to address typical questions such as *Are we doing good enough in assuring our product quality?*,and *How much effort should we spend to increase the quality of our next release?*. (2) The second result is in different types of projects may display a variety of QA activities which depend on the nature of the developer community (e.g. size of the development team, type of project sponsorship, project complexity, and release policies).

Our concept offers to complement the current project monitoring model in OSS project based performance indicators regarding QA processes derived from stakeholder values. However, major challenges for future work were identified: a) how to better formulate such indicators as the basis of meaningful notifications about the status of OSS product quality for different stakeholders, b) how much effort seems reasonable to spend on creating, maintaining and monitoring the indicators in an OSS context; and c) the need for empirical evaluation of the concept using larger set of OSS projects.

References

[1] M. Aberdour. Achieving quality in open-source software. *IEEE Software*, 24, Issue: 1:58–64, 2007.

[2] S. Biffl. *Software Inspection Techniques to Support Project and Quality Management*. Shaker Verlag, 2001.

[3] S. Biffl, A. Aurum, B. Boehm, H. Erdogmus, and P. Gruenbacher, editors. *Value-Based Software Engineering*. Springer Verlag, ISBN:3-540-25993-7, 2005.

[4] A. Capiluppi, P. Lago, and M. Morisio. Characteristics of open source projects. In *Proceeding of the 7th European Conf. Software Maintenance and Reengineering (CSMR 03),IEEE CS Press, p.317330*, 2003.

[5] K. Fogel. *Producing Open Source Software: How to Run a Successful Free Software Project*. O'Reilly, 2005.

[6] F. Manenti, S. Comino, and M. Parisi. From planning to mature: on the determinants of open source take-off. *Discusion Paper, Universita Degli Studi Di Trento*, 2005.

[7] A. Mockus, R. Fielding, and J. Herbsleb. Two case studies of open source software development: Apache and mozilla. *ACM Transactions on Software Engineering and Methodology*, Volume 11 , Issue 3, 2002.

[8] E. Raymond. The cathedral and the bazaar. *http://www.catb.org/esr/writings/cathedral-bazaar/cathedral-bazaar/*, 2003.

[9] S. Valverde, G. Theraulaz, J. Gautrais, V. Fourcassi, and R. Sol. Self-organization patterns in wasp and open source communities,. *IEEE Intelligent Systems,*, Volume: 21, Issue: 2:36– 40, March-April 2006.

[10] D. Wahyudin, A. Schatten, K. Mustofa, S. Biffl, and A. M. Tjoa. Introducing ”health” perspective in open source web-engineering software projects based on project data analysis. In *The Eighth International Conference on Information Integration and Web-based Applications Services*, 2006.