Patch Review Processes in Open Source Software Development Communities: A Comparative Case Study

Jai Asundi, Rajiv Jayant
Information Systems and Operations Management,
School of Management, University of Texas at Dallas,Richardson TX -75080
{asundi, rajiv}@utdallas.edu

Abstract

In spite of the overwhelming success of Free/Open Source Software (F/OSS) like Apache and GNU/Linux, there is a limited understanding of the processes and methodologies that specify this form of software development. In this paper, we examine the process of patch reviews as a proxy for the extent of code-review in F/OSS projects. While existing descriptions of patch review processes are mostly narrative and based on individual experiences, we systematically analyze the email archives of five F/OSS projects to characterize this process. While doing so, we make a distinction between contributions (patches or review comments) by core members and casual contributors to grasp the role of core members in this process. Our results show that while the patch review processes are not exactly identical across various F/OSS projects, the core members across all projects play the vital role of gate-keepers to ensure a high level of review for submitted patches.

1. Introduction

The success of Free and Open Source Software (F/OSS) has grabbed the attention of practitioners and academics alike. While a number of authors [20,3,5] qualitatively describe some of the processes involved in F/OSS, there are few quantitative descriptions or validations (as in [2]) of these processes.

In this paper, we examine the process of patch reviews as a proxy for the extent of code-review in F/OSS projects. Proponents of F/OSS tout the F/OSS process to be superior since "many eyeballs" can potentially read through the code and report defects (Linus' law in [20]). Consequently, a study such as ours is vital if one has to argue (in either way) that there are "enough eyeballs" scrutinizing the source code.

The specific research questions we try to address in this paper are the following: What is the relative proportion of core and non-core members submitting patches for review in F/OSS projects? What is the extent of 'patch review' in F/OSS projects[1]? Are they similar across different projects? Does the extent of review change depending on the type of submitter (non-core versus core member)?

To answer our research questions, we examine five F/OSS projects and collect quantitative information on these projects by examining project artifacts and email archives over a period of time.

In the following section we describe the code review and inspection processes for closed commercial projects. We then compare and contrast these processes with the most commonly found patch-submit-review process for F/OSS projects. We then describe some variants of this process as found in some F/OSS projects. In section 3 we describe our methodology for data collection and analysis. In Section 4 we present our results and offer a discussion of the results. We then conclude in section 5 by summarizing the contribution of this paper and discussing some ideas for future work.

[1] as measured by the number of replies generated by a patch posting

Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS'07)
0-7695-2755-8/07 $20.00 © 2007 IEEE
1530-1605/07 $20.00 © 2007 IEEE

2. Background

2.1. Code Inspections and Code Walkthroughs

The concept of source code inspections was introduced thirty years ago as a cost effective technique to detect software defects [4]. Since then a rich body of literature has examined the processes and components (people, documents and artifacts) that constitute code inspections. For a critique in code inspections, see [26].Recent studies have also identified "enhanced maintainability" as a key benefit of code inspections [22]. While a complete survey of the state of art in code inspections is not the focus of this study, we examine a few that are relevant to this work. For example, Johnson and Tjahjono [23] found that inspection meetings did not find significantly more defects than non meeting based inspection approaches – a finding that is echoed by the previous works like that of Porter and Johnson [24] and Votta [25]. These studies are relevant to this research as the process in F/OSS projects that can be considered equivalent is the patch review process which is non meeting based and carried out asynchronously.

2.2. A Generic Patch Submit-Review Process in F/OSS projects

Fielding [6] was amongst the earliest to describe in detail the patch review process in the Apache server project . The importance of the peer-review process in F/OSS projects has also been outlined by Feller and Fitzgerald [5].

Unlike commercial projects, developers and the reviewers may not physically meet or interact in a synchronous manner. Most patch submit-review processes involve the developer submitting his/her patch to a development mailing list. Submitters are free to submit patches that serve as bug fixes or patches that provide an enhancement. The 'reviewers' then read through the submitted source code and comment on it. F/OSS projects differ from the code-review/inspection process in the sense that the committee/person that 'reviews/inspects' the code is a volunteer unlike in closed/commercial organizations where a reviewer is assigned.

Based on the feedback received by the 'reviewers', the submitter enhances the patch ; once the patch is deemed acceptable, it is committed to the code base by a person with appropriate privileges– typically a core-group member.

The strength of F/OSS development process stems from the fact that contributions can come from any person interested in the project. Also, submission can be 'reviewed' by many people and thus any errors or design flaws that may have crept in can be weeded out. The potential of "many eyeballs" [20] will, as experienced during inspections in closed commercial projects, greatly improve the quality of code by eliminating most of the defects. The downside of this voluntary process is information overload that could lead to one of the following undesirable outcomes:

1. Patch submissions may not get reviewed at all leading to ignored patches
2. Patch submissions may not get a rigorous review leading to a code with a higher number of defects.

3. Methodology

3.1 Choosing Projects for Analysis

In order to analyze the patch review process in F/OSS projects and answer our research questions, we selected projects which we think are, representative of the various strata found in the F/OSS community. Accordingly, we choose Apache, GCC, GDB, Mplayer and Gaim for our study. These projects constitute a diverse mix of system size, targeted end user type, nature of application[2], style of usage[3], and are not contemporaries. Table 1 provides a summary of features and other descriptive information about the projects we have chosen in our study. In the next subsection, we discuss the data collection methodology.

3.2 Data Collection Methodology

3.2.1 Projects with mbox archives. The projects we have chosen for analysis employ different processes for patch submission and subsequent review. In four projects (Apache Httpd, Mplayer , GCC and GDB) contributors are instructed to submit their patches to the development mailing list where they can be discussed. For these four, the email archives of the mailing lists(in mbox format) were downloaded from the respective project websites. Perl scripts were then

[2] projects like Apache, GCC and GDB can be thought of as infrastructure based 'critical applications', while Gaim and Mplayer may be classified as personal entertainment or 'non critical' applications.

[3] The average Apache end-user does not interact as frequently with the application as users of GCC, GDB, Gaim and Mplayer normally do.

used to extract information regarding each email such as the sender's identity, message date, subject lines, header information etc. The subject lines were then parsed for regular expressions that identified patch submissions. While the contributors in the Apache httpd and Mplayer development community were specifically instructed to include the keyword "[PATCH]" in their patch submission subject line, members in other project mailing lists voluntarily used similar easily identifiable regular expressions to distinguish a patch submission [4]. Once we have identified the patch submissions, we use a combination of the message ID and subject-line to identify threaded responses to these patch submissions. The script then counts the number of responses for each patch submission. In addition we also record the person submitting the patch (by type – core/non-core group member) and collect statistics regarding the same.

3.2.2 Projects with Web interfaces. As Gaim is hosted at Sourceforge, it has a web interface, where anyone with a registered Sourceforge account can submit a patch in the patch tracking system for review. Patch discussions, reviews and final decisions regarding the acceptance or rejection of patches are implemented through the Sourceforge web interface. For Gaim, the patch submission database could not downloaded, hence each patch submission was analyzed by querying the patch tracking system via a web-browser and manually collating the information.

For each submitted patch , the patch tracker maintains information about the submitter's name and/or Sourceforge loginID , the date the patch was submitted , the current status of the patch(open / closed) , all the comments which were posted against this patch, the actual patch as a file attachment and the final resolution – i.e., if it was accepted / rejected / previously fixed / out-dated . Each patch submission is indexed by a "RequestID". For each patch, we recorded the submitter of the patch (i.e., Core / Non Core member), the number of core group members and non core group members who responded with a comment, and any other appealing observation applicable to that patch. Patch submissions or review comments posted by anonymous users were counted as contributions/reviews from non-core group members.

[4] To test the soundness of this approach, we randomly sampled 100 subject lines from each project. We found that on an average these were patch submissions 99.5% of the time.

3.2.3 Identifying Core-group members. In most of the projects, we consider the core-group members as those who are identified on the project development pages as "module owners" or individuals with write permissions to the CVS tree. In the Gaim project they are identified as "developers". In almost all F/OSS projects the core-group members are easy to identify from documentation available on the project page. In the Gaim project, one of them being designated as the "Support Manager" who has a special role in the patch submission and review process.

3.3 Discussion of measures and methodology

Due to the manual process employed to collate patch related information in the Gaim project, we assume that the patch has been reviewed if we observe a comment posted against a patch. While counting the number of review comments for a patch, we however do not textually analyze the comments to determine the extent of review or relevance of those comments to the patch. In the other projects, where we analyze the mbox archives, we consider that a patch posting is reviewed if it generates a threaded reply in the email discussion list[5].

Many respondents (especially in the mailing list) use multiple email addresses to post patches/comments. We identified the core developers and their multiple addresses so as to count them only once. Table 1 describes the sample time period for the various projects considered in our study.

Table 1: Project time periods studied

Project	Start Date	Last Date	Num of patch postings
Apache HTTPD	February, 1995	January, 2001	2350
GCC	January, 2000	January, 2005	19487
GDB	April, 2000	January, 2005	7266
Mplayer	April, 2001	January, 2005	2652
Gaim	March, 2000	January, 2005	1848

[5] Given the large number of submissions and responses, we randomly sampled 100 threads in Apache, GCC, GDB and Mplayer. Since gaim involved manual analysis, we sampled about 25 threads in gaim. We find that on an average, the reply was a review 95.1% of the time. This leads us to believe that threaded replies to a patch posting are a reasonable measure of review for the patch.

Table 2: F/OSS projects' descriptive summary

Project Name	Description	No. of Core Developers	Physical Source Lines of Code[6]	End User Type
Apache HTTPD	A popular HTTPd webserver that delivers static as well as dynamic web pages to browsers	51	194K	Technically skilled user, usually a Webmaster
GCC	The GNU Compiler Collection - provides a suite of compilers for C, C++, Ada, Java, Fortran etc.	193	1967K	Software Developers
GDB	A debugger for C, C++, Objective C , Pascal and some other languages.	44	1282K	Software Developers
Mplayer	A video player for GNU/Linux that can play most of the popular video formats – MPEG, AVI, Ogg, ASF,QT etc.	58	509k	Casual Users
Gaim	A multi-protocol chat client .	15	155K	Casual Users

4. Results and Discussion

4.1 Observations on the patch-submit-review process

The generic patch-submit-review process as described in Section 2 is observed in most of the projects in our study. For example, initially the Apache httpd project had a voting system where code changes would be allowed if 3 other developers gave positive(+1) votes and there were no negative (-1) votes for that change [6]. Subsequently, the process evolved to a less bureaucratic form, but patch submissions by non-core members still had to be reviewed before they could be checked in. We thus observe that core group members act as "gatekeepers" ensuring that the code received from the community is scrutinized before being checked in. Similar processes are found in other projects which use the mailing list for patch submissions and review (GCC, GDB, Mplayer).

A variant of the generic process described in section 2 is in projects like that of Gaim (or Mozilla), where all patch submission and review is carried out through a web-interface. The project uses the patch tracking tool available at Sourceforge to manage the submission and review of patches. Submitters upload patches and request a review. The project documents acknowledge that requests for review may not receive a response in a week's time in which case they expect the patch submitter to either approach another known expert for that module, or ask around the community for help in getting the patch reviewed. This process seems to be closer to the known process of code inspections/walkthroughs in a closed commercial software project where senior members of the project are requested to review a junior developer's code. Gaim also lists a person in the core group as a "Support Manager". This individual seems to play a key role since within the time period analyzed in our study, 54% of all patches posted (1005 of 1848) were first evaluated by the "Support Manager" who either assigned the patch to an appropriate core-group member or took a decision to accept/reject the patch.

[6] Provided for reference purpose only. It reflects the size of the latest stable release. Data generated using 'SLOCCount' by Wheeler (2000-2001)

Table 3: Summary of Results

	Apache httpd	GCC	GDB	Mplayer	GAIM
Total number of patch review postings	2350	19487	7266	2652	1848
Percentage of patch postings by Core Group Members	73.95%	80.12%	71.08%	24.66%	12.44%
Percentage of patches that were submitted by or reviewed by a Core group member	88.72%	90.85%	91.91%	75.76%	71.81%
Percentage of patches submitted by Core Group Members that did not generate a reply	32.85%	43.81%	45.18%	26.91%	53.91%
Percentage of patches submitted by Non Core group members that did not generate a reply	43.3%	46.06%	27.98%	32.18%	32.22%
Average number of respondents per patch posting	1.81* (2.10)	1.26* (2.34)	1.31* (1.98)	1.77* (2.58)	0.94 (1.13)
Average number of Core group respondents per patch posting	1.62* (1.86)	1.09* (1.99)	1.04* (1.59)	1.01* (1.70)	0.74 (0.68)
Average number of replies per patch posting	2.92* (4.69)	2.14* (5.05)	2.45* (4.51)	3.20* (5.92)	NA
Average number of replies by Core group Members per patch posting	2.62* (4.27)	1.90* (4.63)	1.92* (3.67)	1.80* (3.52)	NA

If we focus on recent (past 2.5 years) patch postings, we find that the "Support Manager" was the first to act (i.e accept/reject the patch or delegate for review) in 70.9% (969 of 1366) of patches posted in this duration. We can consider this imposed structure as an effort by the Gaim developers to streamline the review process and ensure the review of patch postings.

In cases where security holes are being patched, the review process is carried out only by a select group of core members and is not discussed on a public forum and committed to code base as soon as the patch is considered effective. These are the situations where the patch submit-review process is not observable.

4.2 Quantitative Results

The summary result of our analysis is in Table 3. We observe (from row 2) that the percentage of patch submissions by core group members in some of the old infrastructure related projects such as Apache-httpd, GCC and GDB is higher than the user utility projects like MPlayer and Gaim. From row 3 of Table 3, we find that the percentage of patches that involved a core group member i.e., the patch was either submitted by a core-group member or was reviewed by a core group member is greater than 70% for all projects and as high as ~90% for the three infrastructure related projects. This relatively high percentage indicates that the core group has a high level of involvement in the patch submission and review process and act essentially as gatekeepers to the code-base. From rows 4 and 5 of Table 3, we observe that the percentage of patches submitted that go unreviewed does not show a clear trend with respect to the type of submitter (core or non-core). For example, while more than half (53.91%) of the patches submitted by Gaim core developers do not get any review, only a quarter (26.91%) of the patches submitted by Mplayer core developers go unreviewed. To analyze if the proportion of submitted patches that received at least one review is the same across projects, we conducted a Chi-squared test for differences in probabilities using a r X c contingency table approach. The null hypothesis is that the

proportion of patches reviewed is same across all projects. The test result yielded a p-value of less than 0.0001 implying that the proportion of patches that received reviews did differ significantly across different projects. Finally, the average numbers described in the last four rows describes the extent of response generated by the entire community as well as the core-group members due to a patch submission. We observe that for four out of five projects on an average, a patch posting will get at least one core member responding to a patch submission.

4.3 Discussion

Based on the results, we see that core members account for a greater proportion of submitted patches in infrastructure projects like Apache, GCC and GDB. This could be due to a couple of reasons. First, it is possible that these projects require a specialized skill-set (like assembly language, compiler design etc) which may be hard to find amongst a more general user community. Second, it may be the case that core-members in non critical application projects like Mplayer and Gaim check-in their patches directly into the code-base rather than submit it for peer review. While on an average each patch submission evokes response from at least one core member to respond, in many cases we observe that the core member(s) will respond multiple times to create a cycle of patch-review-improve. The r X c contingency table analysis shows us that the extent of review differs across projects. The reasons for this could again be attributed to the specialized skills required to contribute and review patches.

The percentage of patches that goes unreviewed merits further discussion. Usually patches submitted by non-core members go un-reviewed because the issue is being addressed in some other thread or has already been resolved. In some cases the patch submissions are not reviewed because they do not adhere to the format for patch submission as specified in the project documents. For example, in the Apache HTTPD project we note that of the 200 identified individuals who submitted patches; only 33 did not get a response. Further analysis showed that 11 of them did not post the patch in the correct format and 10 were responded to in different threads. Based on this observation, we could assume that the percentage numbers we have presented in Table 3 is a lower bound on the extent of review in these projects. For those patches that were submitted by core group members which go unreviewed, only an examination of the source code will help determine if these patches were eventually applied in spite of the lack of review.

5. Conclusions and Future Work

The inspection of source code has always been considered to be effective in detecting defects in a software program. The patch review process can be considered to be the F/OSS equivalent process for the code inspection/walkthroughs that occur in closed commercial projects. We find that F/OSS projects vary in terms of their patch review processes and characteristics. One possible explanation for the variation in the quantitative measures could be explained by the culture these projects inculcate in their participants. We notice that the Apache HTTPD, GCC and GDB projects show some specific characteristic such as a high level of core-group involvement and review percentage. This may be due to the fact these projects are relatively older and also be attracting some of the (relatively) older developers that adhere to certain formal processes. Variations may also be due to the inherent complexity of the product that may demand a more rigorous review process. Availability of documentation, coding styles and project popularity could be other factors that influence the participation of non-core members in the review process

This paper contributes to the existing empirical literature on F/OSS projects. We believe that this study is amongst the first to empirically examine the issue of peer-review in F/OSS projects. We have described some innovative measures that can be used to monitor the extent of peer review and community participation in F/OSS projects. We believe that these quantitative measures will help us form a basis for better understanding of F/OSS projects and a means to monitor and control the quality of the software product developed.

This research leads us to some interesting research questions like what determines the extent of patch review in F/OSS projects?, Are there identifiable characteristics that are likely to lead to a high (or low) level of review? What proportion of unreviwed patches make it to the code base? How does the extent of review impact the F/OSS project as a whole? Are there significant implications on product quality, project success and other product measures? We plan to explore these questions in future work.

6. References

[1] Ackerman, F., L.S. Buchwald and F.H. Lewski, 1986, Software inspections: An effective verification process, IEEE Software, 6(3), May, pp. 31-36.

[2] Crowston, K. and Scozzi, B., 2004, Coordination practices with FLOSS development teams: The bug fixing process, Computer Supported Activity Coordination 2004: 21-30

[3] DiBona, C. Ockman, S. and Stone, M. (eds.) 1999, Open Sources: Voices from the Open Source Revolution, O'Reilly, Sebastopol, CA

[4] Fagan, M. E. 1976, Design and code inspections to reduce errors in program development, IBM Systems Journal, 15(3):182-210

[5] Feller, J. and Fitzgerald, B. 2002. Understanding Open Source Software Development Addison Wesley, London, England.

[6] Fielding, R.T. 1999. Shared Leadership in the Apache Project, Communications of the ACM, vol. 42, no. 4, 42-43.

[7] Grady, Robert B. and Thomas van Slack, 1994, Key lessons in achieving widespread inspection use, IEEE Software, 11(4):46-57.

[8] Hahsler, M. and Koch, S., 2005, Discussion of a Large-scale Open Source Data Collection Methodology, Proceedings of the 38th Hawaii International Conference on System Sciences(HICSS-38), Big Island, Hawaii.

[9] von Hippel, E. 1988. Sources of Innovation, Oxford University Press, New York.

[10] Jalote, P. 1997. An Integrated Approach to Software Engineering, 2nd Ed., Springer-Verlag, New York.

[11] Jones, C., 1977(?), Programmer quality and programmer productivity, Technical Report TR-02.764, Yorktown Heights, NY:IBM.

[12] Jones, C. 1991, Applied Software Measurement, McGraw Hill, New York.

[13] Jorgenson, N. 2001, Putting it all in the trunk: Incremental software development in the FreeBSD open source project, Information Systems Journal, 11, 4.

[14] Koch, S., 2003, Effort Estimation in Open Source Software Development: A Case Study, in Proceedings of the 2003 IRMA International Conference, Philadelphia, PA, pp.859-61

[15] von Krogh, G., Spaeth, S. and Lakhani, K.R., 2003. Community, Joining, and Specialization in Open Source Software Innovation: A Case Study. Research Policy 32 (7):1217-1241.

[16] Lerner, J., Tirole, J. 2002. Some Simple Economics of Open Source, Journal of Industrial Economics, 52, 197-234

[17] Mockus, A., Fielding, R.T., Herbsleb, J. 2000. A Case Study of Open Source Software Development: The Apache Server, Proceedings of the 22nd International Conference on Software Engineering, Limerick, Ireland, pp. 263-272

[18] Mockus, A., Fielding, R.T., Herbsleb, J. 2002. Two case studies of open source software development: Apache and Mozilla, ACM Transactions on Software Engineering and Methodology, vol. 11, No. 3, 309-346

[19] Pfleeger, S.L. 2001. Software Engineering: Theory and Practice, 2nd Ed., Prentice Hall, New Jersey.

[20] Raymond, E. 1999. The Cathedral and the Bazaar, O'Reilly, Sebastopol, CA.

[21] Wheeler, D., "More Than a Gigabuck: Estimating GNU/Linux's Size, June 30, 2001 (updated July 29, 2002) , Version 1.07 , http://www.dwheeler.com/sloc/redhat71-v1/redhat71sloc.html accessed on February 28th 2005

[22] Siy and Votta, Does the modern code inspection have value?,Proceedings of the IEEE International Conference on Software Maintenance, 2001. pp 281-289

[23] Johnson, P. M. and Tjahjono, D. Does Every Inspection Really Need a Meeting?. Empirical Softw. Engg. 3, 1 (Jul. 1998), 9-35

[24] Porter, A. A. and Johnson, P. M. 1997. Assessing Software Review Meetings: Results of a Comparative Analysis of Two Experimental Studies. *IEEE Trans. Softw. Eng.* 23, 3 (Mar. 1997), 129-145.

[25] Votta, L. G. 1993. Does every inspection need a meeting?. In *Proceedings of the 1st ACM SIGSOFT Symposium on Foundations of Software Engineering* (Los Angeles, California, United States, December 08 - 10, 1993). D. Notkin, Ed. SIGSOFT '93. ACM Press, New York, NY, 107-114.

[26] Glass, R. L. 1999. Practical programmer: inspections—some surprising findings. Commun. ACM 42, 4 (Apr. 1999), 17-19.