On the Need for OSS Patch Contribution Tools

Bhuricha Deen Sethanandha[1], Bart Massey[1], and William Jones[2]
1 Department of Computer Science, Portland State University
1900 SW 4th Avenue, Portland, OR 97201, USA
{bhuricha, bart}@cs.pdx.edu
2 The Information School, University of Washington
Mary Gates Hall, Ste 370, Seattle, WA 98195, USA
williamj@uw.edu

Abstract. Open Source Software (OSS) projects and systems have become significant parts of the software economy. The sustainability of an OSS project depends largely on community contributions. The patch contribution process is important to OSS projects. Nevertheless, there are several issues negatively impacting patch contribution in mature OSS projects. These issues can be addressed by improving tools to support the patch contribution process.

1 Introduction

The number of Open Source Software (OSS) projects and the size of OSS systems have been growing at an exponential rate [1], yet the majority of them do not have enough members to become sustainable [2]. The sustainability of OSS projects depends largely on community contributions [3]. However, it takes a significant amount of time and effort for external developers to contribute to OSS projects [4]. This is especially true for patch (source code and document change) contributions. It takes a significant amount of time to learn the technical and social aspects of the project, and even more time to gain trust by demonstrating skills and accumulating reputation, [4]. Moreover, in order to contribute changes, external contributors have to go through a patch contribution process that has a number of potential barriers.

The OSS patch contribution process (PCP) [5] is important for the sustainability of OSS projects because it enables learning and knowledge transfer in software projects [6] and provides an opportunity for recruiting potential developers into OSS projects [7]. By improving the OSS PCP, more patch contributors can be processed in a timely manner and more contributors can be motivated to contribute more. In this paper, we analyze key issues with the PCP. We conclude by providing suggestions for improved PCP tools that address these issues.

2 Key Patch Contribution Process Issues

Despite the long history of open source the PCP remains somewhat immature and, until recently, poorly studied. Using the data collected from ten OSS projects [5], we analyze common issues among them and identify and explain how existing tools contribute to these issues. In this section, we focus on identifying problems causes by the tools that are used to support patch review and discovery activities.

Patch contribution is time-consuming and slow

The PCP is time consuming and slow because it consists of several activities: creation, publication, discovery, review and application. Many of these activities involve several people including patch contributors, reviewers, peer developers and committers [5]. Moreover, this process is performed asynchronously. Any process of many steps that is executed serially and cooperatively is likely to be difficult and take a long time; the PCP is no exception.

The amount of time that it takes to complete the process depends on many factors such as contributor experience, the capability of tools and the quality of patches. Processing a patch inevitably takes hours and often takes weeks [7, 8, 9]. Although patch review is an important defect detection mechanism, in some projects only patches created by new contributors are required to be reviewed prior to being committed; a policy intended to reduce the delay and complexity for experienced contributors [8]. Literature has identified the slow nature of the PCP as an issue, but studies have yet to address how current patch review tools address this issue.

The PCP can also be time consuming because it requires contributors and reviewers to access information from various sources in order to complete the process. These sources may include email, project web pages, revision control systems and bug tracking systems. It is difficult to get all the information needed to learn about the process and culture because the information is scattered across sources. For example, to find the potential reviewers, newcomers may have to look up the project web pages, bug database and revision log to determine who should be included in patch requests. There is a need for a tool that provides social information as newcomers learn about the technical aspects of the OSS projects.

Patches can be reviewed using email, the issue tracking system or a dedicated patch review system. Information needed in order to review patches effectively is also found in many places. Reviewing patches in email requires both fresh knowledge of the code affected by patches and also guidelines such as coding standards. Patches in email usually describe specific changes to a section of code, but do not provide context in the form of a full source file. This approach works reasonably well for experienced developers who know a lot about the area where the code patch applies and are fluent in reading patches, but less well for inexperienced reviewers. The approach thus limits the possibility that peer developers can participate in patch review. Advanced patch-review skills require a lot of time and experience to acquire. Recently, web-based patch review systems such as Review

Board[1] and Gerrit[2] have been developed to address problems introduced by patch review using email. These tools have many nice capabilities such as side-by-side code comparison, inline comments, and integration with revision control systems. However, neither of these tools is widely adopted by OSS projects, and there is as yet no evidence that these tools are better than mailing lists. Nevertheless, they represent a potentially important step toward an improved PCP.

More in-depth analysis of the existing patch review tools are needed in order to improve the patch review tools. A better patch review tool can provide information needed when reviewers review patches, thus reducing the time required to complete the task.

Patches are often lost or ignored, and under-reviewed

Although OSS projects encourage patch contribution, they have problems handling the amount of patches they receive. The Apache project, where a peer review process is mandatory, 23% of submitted patches are ignored and 8% of the commits are un-reviewed [8]. In many projects, a significant number of patches did not receive responses [10]. Patches might be ignored unintentionally when the reviewers are too busy with other OSS development tasks. Sometimes they can be ignored because of the poor quality of the submitted patches [6]. Moreover, in many OSS projects, only a small group of people participate in patch review [6, 8]. The average number of reviewer responses per patch posting is less than two and the majority of the patches are reviewed by one person [8, 10].

Mailing lists (ML) and issue tracking systems (ITS) are the most commonly used tools for publishing patches. As the volume of email or issues increases, it becomes harder to keep track of patches, which may then be lost or ignored. Mailing lists have the potential to deliver patches to more potential reviewers, but in fact only active members who constantly monitor the list may discover them. Patch tracking systems such as Patchwork[3] and CommitFest[4], have recently been developed to make it easier for patch contributors and reviewers to track the status of patches reviewed over a mailing list. Yet, we observe a long list of unattended patches on the Linux Kernel patch list. Many OSS projects require patch contributors to publish patches to their ITS, which is not designed to support the PCP. Therefore, it is difficult to find un-reviewed patches and to track the process of patches. This is a challenge for projects, such as Mozilla and Eclipse, that use ITSs. Due to the search limitations of the ITSs, reviewers also have difficulty discovering patches that are not assigned to them. Hence, these OSS projects rely on patch contributors to specify potential reviewers. Contributors must spend time learning about the projects in order to find the reviewers. Although the Drupal project has an ITS that provides the list of unattended patches, they still have problems getting reviewers to review them. None of the PTSs or ITSs we reviewed provides an effective way for reviewers to discover

[1] The Review Board project http://www.reviewboard.org/
[2] Gerrit Code Review http://code.google.com/p/gerrit/
[3] Patchwork – Web-based patch tracking system http://ozlabs.org/~jk/projects/patchwork/
[4] PostgreSQL CommitFest https://commitfest.postgresql.org/

patches that were not assigned to them. Consequently, the majority of patches are ignored or under-reviewed. Better support for patch discovery and awareness is needed to solve these problems. Making patches easily discoverable should increase the number of peer developers reviewing patches, which will reduce the number of ignored patches and increase the number of reviewers per patch.

Improving Tool Support for OSS Patch Contribution Process

The PCP often involves many tools that patch contributors and reviewers are required to switch between in order to complete the process. Integrated development environments (IDEs) are integrated with issue tracking systems, web browser, and revision control systems in order to support software development tasks. However, they do not yet support the PCP. The cost of context switching can be largely eliminated by changing the IDE to support PCP related tasks, which will increase the productivity of patch contributors and reviewers. For example, to publish a patch, the IDE can automatically create a patch from the changes to the code modified by the contributor, and then automatically analyze the patch based on OSS project coding standards. It can also guide the contributor through the patch publication procedure for the target OSS project without leaving the IDE. Reviewers receive notification on their task list within the IDE that a new patch needs to be reviewed. Reviewers can then perform the review task within the IDE and provide feedback. Once the patch is approved, it is automatically committed to the code repository by the IDE.

In order to improve tools that support *patch review*, the ideal tool has the following features:

- *Provides recommendations for relevant information that reviewers need to complete the review.* For example, it should provide a link to the source files modified by the patch, a link to the coding standard, and a checklist describing what reviewers should look for. This should reduce the amount of time that reviewers need to review patches.
- *Provides different feedback options.* Most patch review tools require the reviewer to provide textual feedback, which is time-consuming for reviewers. A feedback system that lets reviewers go through a set of questions or a simple rating scale may make providing feedback easier and faster, which may encourage more peer developer to review.
- *Provides instant feedback for issues that can be recognized automatically.* Trivial errors, such as incorrect coding style, can then be handled without requiring reviewer feedback. If the tool can handle most of the preventable reasons for revision, it can reduce the number of resubmissions and allow reviewers to focus on implementation and design issues. This should reduce the amount of time that takes to resolve patches, and increase the quality of patches.

In order to improve tools that support *patch discovery*, the ideal tool has the following features:

- *Provides a better view to represent patch status.* Kanban board visualization and a cumulative flow diagram can help increase project visibility and let

the team visually track progress and identify bottlenecks [11]. A Kanban board is a board that is divided into different columns representing the status of tasks. Task cards are placed in these columns based on their status. Patches can be displayed as tasks with their status displayed: need review, reviewing, reviewed, rejected, etc. The Kanban board lets reviewers and contributors detect the number of patches that are queued under each status, and work together to reduce the bottleneck.

- *Provides an ability to tie patches to a specific location in the code base*. There are many benefits to using this information. This capability enables developers and reviewers to discover patches and easily search for patches related to the code they are working on. For reviewers, they are likely to review patches related to the code they have experience with. The ideal person would be the person who wrote the code modified by a patch. Newcomers need to be aware of patches related to patches they are working on, which is hard to do using existing tools. The following capabilities can be developed using location tag information:
  - Web-based code browsers can provide patch information in addition to source code information.
  - A recommendation system that is part of IDEs can provide information about patches based on the part of code developers are working on.

Conclusion

The PCP is important to sustainable OSS projects. Although there are many issues that become barriers to contributions, they can be addressed by improving the existing tools. We propose several tool improvements: incorporating patch review and notification in an IDE, a patch-aware code browser, and Kanban board visualization. Further study is required in order to design and implement such tools in order to increase process efficiency and patch discoverability.

References

[1] A. Deshpande and D. Riehle, "The Total Growth of Open Source," *Open Source Development, Communities and Quality*, 2008, pp. 197-209.

[2] K. Crowston, J. Howison, and H. Annabi, "Information systems success in free and open source software development: theory and measures," *Software Process: Improvement and Practice*, vol. 11, 2006, pp. 123-148.

[3] C. Wu, J.H. Gerlach, and C.E. Young, "An empirical analysis of open source software developers' motivations and continuance intentions," *Information & Management*, vol. 44, Apr. 2007, pp. 253-262.

[4] G. von Krogh, S. Spaeth, and K.R. Lakhani, "Community, joining, and specialization in open source software innovation: a case study," *Research*

Policy, vol. 32, Jul. 2003, pp. 1217-1241.

[5] B.D. Sethanandha, B. Massey, and W. Jones, "Managing Open Source Contributions For Software Project Sustainability," *Management of Engineering & Technology, 2010. PICMET 2010. Portland International Conference on*, Bangkok, Thailand: 2010. (to appear)

[6] M. Nurolahzade, S.M. Nasehi, S.H. Khandkar, and S. Rawal, "The role of patch review in software evolution: an analysis of the mozilla firefox," *Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops*, Amsterdam, The Netherlands: ACM, 2009, pp. 9-18.

[7] C. Jensen and W. Scacchi, "Role Migration and Advancement Processes in OSSD Projects: A Comparative Case Study," *Proceedings of the 29th international conference on Software Engineering*, IEEE Computer Society, 2007, pp. 364-374.

[8] P.C. Rigby, D.M. German, and M. Storey, "Open source software peer review practices: a case study of the apache server," *Proceedings of the 30th international conference on Software engineering*, Leipzig, Germany: ACM, 2008, pp. 541-550.

[9] P. Weißgerber, D. Neu, and S. Diehl, "Small patches get in!," *Proceedings of the 2008 international working conference on Mining software repositories*, Leipzig, Germany: ACM, 2008, pp. 67-76.

[10] J. Asundi and R. Jayant, "Patch Review Processes in Open Source Software Development Communities: A Comparative Case Study," *Proceedings of the 40th Annual Hawaii International Conference on System Sciences*, IEEE Computer Society, 2007, p. 166c.

[11] M. Poppendieck and T. Poppendieck, *Lean software development : an agile toolkit*, Boston: Addison-Wesley, 2003, p. 76.