Managing Open Source Contributions for Software Project Sustainability

Bhuricha Deen Sethanandha[1], Bart Massey[1], William Jones[2]
[1]Department of Computer Science, Portland State University, Portland, USA
[2]The Information School, University of Washington, Seattle, USA

Abstract— The use of open source software has become a part of accepted business strategies. A primary strength of open source software is its leverage of outside innovation. All are free to take open source software and use it, evaluate it, repair it, and add new capabilities. One perceived risk of using open source software components in commercial systems is open source project sustainability. It would be expensive for the project supporting a critical open source component to fail midway through the life cycle of a commercial product. Many commercial organizations reduce this risk by contributing to the open source projects that they use. However, the "contribution barrier" for successful open source software projects is high, especially for commercial contributors. This barrier has technical and social components, both of which are exacerbated by minimal attention paid to good management practices. This paper proposes a process for managing open source software "patch" (source code and documentation change) contributions. By observation and by examination of current literature we identify key practices for patch creation, publication, discovery, review, and, application. An improved patch contribution process will lower the contribution barrier, helping to improve the sustainability of critical open source projects.

I. INTRODUCTION

The number of Open source software (OSS) projects and the size of OSS systems has been growing at an exponential rate [9], becoming a significant part of the software economy. The success of an OSS project depends largely on the quality of the OSS system and the community surrounding it. Core developers are considered the elite within the community because their contribution directly affects the quality and growth of the OSS system. Becoming a core developer comes at high cost, particularly in mature OSS projects. It takes a significant amount of time to learn both technical and social aspects of the projects, and an additional amount of time to gain trust by demonstrating skills and accumulating reputation [13]. These costs act as a barrier to contribution to OSS projects. As OSS projects have become popular, more developers have become interested in participating in OSS projects. Some of these are volunteers [11], and some are sponsored by public and private organizations [3, 15, 30]. Indeed, companies and governments are now using open source as a critical strategic tool [6, 26] . Regardless of their background, developers typically start by contributing patches [30]. Patches are sets of modifications to an open source project's source code. The patches have to go through various stages before being included in public releases of OSS systems.

A process that has the goal of incorporating patches into an OSS project's source code or documents is a *patch contribution process*. The patch contribution process affects both the quality of the OSS system and the growth of the developer community. There are three main reasons that the patch contribution process is an important activity for OSS projects.

1. It is a primary quality assurance mechanism for OSS systems, especially for contributions from new contributors. A good patch contribution process includes patch review activity that prevents bad patches from getting into the source code. As a result, the quality of OSS systems is comparable to their commercial counterparts [17].

2. The patch contribution process also enables learning and knowledge transfer in software projects [19]. A patch contribution in a form of bug (programming error) fixes is one of the most common contributions from contributors who later become part of the developer community [13].

3. The patch contribution process provides an opportunity for recruiting potential developers into OSS projects. Patch contributors are able to gain more central roles in the open source projects, when their skills and dedication are recognized by community leaders [12]. Despite the importance of the patch contributions, it is only recently that it has been explored by researchers [1, 4, 19, 25]. Most of these works focus on the patch review activity.

Drawing on data collected from ten OSS projects and literature, we propose a patch contribution process model. This model encompasses patch *creation, publication, discovery, review* and *release.* Our work comprises two main contributions. First, it provides a transparent generic model of the OSS contribution process. Second, it provides an example of recommended practices drawn from several successful OSS projects. This paper provides a foundation for understanding how companies can successfully gain competitive advantages from contributing to or managing OSS projects [10, 27] even given the extra difficulties of corporate contribution [18].

The paper continues in four sections. Section 2 reviews literature on OSS software development and challenges in OSS contribution process. Section 3 describes our methodology and data sources. Section 4 presents the conceptual model of OSS contribution process. Section 5 concludes the paper by discussing use of the conceptual model and future directions.

II. LITERATURE REVIEW

A. *Open Source Software Development*

Mockus (2000), et al were among the earliest who conducted research in open source software (OSS) development. They examined the development process of Apache by analyzing email archives of source code changes and problem reports. They suggested that a hybrid form of OSS and commercial techniques could lead to better software development processes. The authors describe the key characteristics of OSS projects. First, OSS projects consist of large number of volunteers [21]. Second, work is not assigned; anyone can choose task they want to do. The OSS development teams generally self-organize their work, and self-assignment is the most common mechanism for task management [7]. Third, there is neither explicit system design nor detailed design [10]. The requirements of OSS systems are usually quite informal [27]. Fourth, there is no project plan, schedule or list of deliverables [17]. Despite the lack of traditional coordination mechanism such as plans, defined processes and design documentation, the quality of OSS systems is comparable to their commercial counterparts [29]. The quality is attributed by the review process done by a large number of contributors and the expertise and passion of the developer on the work they choose to do [14].

There are many open source strategies that organizations use in order to strengthen their businesses. Commercial firms such as Google, Hewlett-Packard IBM, and Sun Microsystems release their some of products under open source licenses. These companies gain more profits from on the complementary products as their open source products become widely adopted [10, 33]. Smaller companies such as Ubuntu and Red Hat also benefit from open source as a strategy. Companies often seek to become part of the communities by using their resources to contribute to the OSS projects [8], positioning their contributors to become community leaders. The public sector has also adopted OSS in order to enable collaboration between different organizations, reduce the risk of vendor lock-in and enable cost savings and innovation [22, 28].

B. *Issues in Open Source Patch Contribution*

Most open source software (OSS) projects accept code contributions in the form of patches. Patches are sets of modifications to the existing codebase of open source projects. The patch review process is one of the most important activities in OSS development because it ensures that patches meet the project standards [17, 25]. The asynchronous nature of OSS development makes it practical to use this quality assurance mechanism [20]. Thus, it has become the primary mechanism in many open source projects where automated testing is not applicable [23]. In addition to its importance for software quality, patch review also enables learning and knowledge transfer in software projects [19]. The patch review process provides a means for patch contributors to demonstrate their technical skills and commitment to the core developers. Patch contributors can gain more central roles in the open source projects if their skills and dedication are recognized by community leaders [12]. Therefore, this process is crucial the sustainability of open source projects [5].

Nevertheless, there are four main challenges and issues in the study of OSS patch review that become significant barriers to OSS contributions.

1. Patch review varies between OSS projects [1, 24]. The differences in processes and tools can cause confusion and require more time to learn when contributors participate in multiple projects. Asundi conducted a study on five OSS projects and provides a textual description of a generic patch submit-review process. However, this work is not sufficient to understand and identify key contribution process elements.

2. Patch review is time consuming and slow. For example, the Apache web server project was required every patch to be reviewed before being applied to the project code repository, but allowed committers to apply the patches to the project source code directly prior to the review [25].

3. Patches are often lost and ignored. The number of un-reviewed patches in an open source project can range from 27% to 54% of the submitted patches [1]. In the Apache project, where a peer review process is mandatory, 23% of submitted patches are ignored and 8% of the commits were left un-reviewed [25].

4. The majority of patches are rejected. More than half of submitted patches are rejected, and there are significantly more patch contributors than reviewers [32]. The common reasons for patches to be rejected relate to implementation and design issues [19].

III. METHODOLOGY AND DATA SOURCES

We attempt to identify a generic OSS patch contribution process by reviewing the processes of a wide range of OSS projects. Using similar criteria as Asundi et al. [1], we selected projects to study from among the most popular active projects from the ohloh.net website[1]. We grouped these projects by application domain, considering e.g. operating systems, database systems and web browsers. From each group we selected only one candidate. We took care to insure that selected projects should not be under the control of the same supporting foundation or company. We also used other criteria such as project size, number of contributors, tenure and supporting tools. Eventually, we chose the following projects as exemplars: Android, Apache web server, Drupal, Eclipse Mylyn, KOrganizer, Linux Kernel, Mozilla Firefox, OpenOffice.org, PostgreSQL and X.org. Table I. provides brief summary of project characteristics.

[1] For the list of popular projects see http://www.ohloh.net/p, January 2010

TABLE I. *OSS* PROJECTS' *DESCRIPTIVE SUMMARY*

Project Name	End User Type	Tenure (Years)	Size (Line of Code)	Number of Contributors	Number of Commits
Android	Developers, Casual Users	1.5	5,556,884	331	5,997
Apache web server	Webmaster	14	675,105	102	37,306
Drupal (Core)	Casual Users, Webmaster	11	116,274	31	11,364
Eclipse Mylyn	Developers	5	905,596	14	9,152
KOrganizer	Casual Users	5	340,768	105	2,786
Linux Kernel 2.6	Expert Users, Developers	8	8,216,262	6,009	172,334
Mozilla Firefox	Casual Users	8	63,714	156	7,042
OpenOffice	Casual Users	10	23,195,306	481	786,169
PostgreSQL	Developers, System Admin	14	569,381	47	28,311
X.org	Expert Users, Developers	10	1,641,722	752	39,845

Data on the projects under study was obtained from OSS project websites. We analyzed and compared the documentation that describes their patch contribution processes to identify key process components. Although some sort of patch contribution process is common, it often is not well defined in project documents. Instead, information related to the patch contribution process is scattered across multiple documents. The description of major process steps also typically differs in level of detail both within and across projects. Some projects provide detailed information, while some provide a short workflow description. We aggregate information from these projects to create a complete description of the contribution process. We also use the proposed model to verify against the documentation to make sure that these steps exist in the project.

IV. THE OSS PATCH CONTRIBUTION PROCESS

Although, the actual patch contribution processes of the open source projects under study differed, we found many common elements among them. This finding is similar to other studies [1, 24]. In addition to their findings, we found that the conceptual level of the contribution process is similar across projects; the software work products, roles and activities involved in the process tended to be similar. In this section, we follow Lonchamp's process modeling approach and describe the process using the Software Process Engineering Meta-model (SPEM) [16].

A. *Software Work Products and Tools*

The most important work product of an OSS project is the project *codebase*. The codebase is the whole collection of source code used to build an OSS system. The project's codebase is typically stored in a source control repository. Source control systems used in the project under study included CVS, Subversion and Git. In general, developers modify a local copy of the project source code. This local copy is called a working copy, and resides on a developer's computer. When they want to make modifications to the project codebase, developers need to describe the changes in the form of patches.

A *patch* is the central work product in the modification of OSS. The patch is a text file that describes a number of modifications made to one or more source code files within a *working copy of a codebase*. Patches are created by a program called *diff*, or directly by a source control system. A patch contains information such as the name of the patch author and patch reviewer, the file being modified, and patch *hunks*, which are segments of the source files that have been modified. Each hunk represents changes in from of deleted lines follow by added lines.

```
Index: sw/source/core/text/txtfrm.cxx
===================================================
================
---                   sw/source/core/text/txtfrm.cxx
         (revision 274676)
+++ sw/source/core/text/txtfrm.cxx         (working
copy)
@@ -651,8 +651,8 @@
 xub_StrLen   SwTxtFrm::FindBrk(   const   XubString
&rTxt,const  xub_StrLen  nStart,  const  xub_StrLen
nEnd ) const
 {
-         xub_StrLen nFound = nStart;
-         const  xub_StrLen  nEndLine  =  Min(  nEnd,
rTxt.Len() );
+         unsigned long int nFound = nStart;
+         const unsigned  long  int  nEndLine  =  Min(
nEnd, rTxt.Len() );

          //  Wir  ueberlesen  erst  alle  Blanks  am
Anfang der Zeile (vgl. Bug 2235).
          while(  nFound  <=  nEndLine  &&  '  '  ==
rTxt.GetChar( nFound ) )
@@ -665,7 +665,7 @@
          while(  nFound  <=  nEndLine  &&  '  '  !=
rTxt.GetChar( nFound ) )
                    ++nFound;

-         return nFound;
+         return (xub_StrLen)nFound;
 }
```

Figure 1. A patch in the unified format

Fig. 1 above is an example of a patch in the *unified format* that fixes problem #104291 of OpenOffice.org. The first four lines are called the patch header. The lines preceded by "@@" are the beginning of the patch hunk, the lines preceded by a space are context lines, the lines preceded by a minus sign are deleted lines, and the lines preceded by a plus sign are added lines . The context lines are surrounding lines that are unchanged by the patch hunk. The context provides more information to help developers understand the changed lines. It also helps to ensure that a patch will be properly applied to the project's source code.

There are many purposes for a contributed patch, such as bug fixing, code maintenance, enhancement or creation of a new feature. The size of the patch can range from one line to hundreds of line of code, and it may modify several files. The complexity of patches varies widely. Not all patches apply to the project's source code: changes to documentation; the build system and the like may also be recorded as patches.

A patch can be published independently via email, as part of an *issue report* in an issue tracking system or as a patch item in a patch tracking system. An *issue report* is a work product that contains information related to software issues. An issue can be a bug, a feature, an enhancement or a task. Issue reports are stored and managed by an issue tracking system or bug tracking system. The issue tracking system[2] allows patches related to an issue report to be attached to the report. Comments on patches and the issue report itself are also supported. The number of issue reports in the issue tracking system may be high, but only a fraction of these contain a patch. A patch tracking system[3] may also be used instead of an issue tracking system to publish, manage and track patches for an OSS project. The common features are an ability to list patches based on their status, the ability to add comments related to patches, and the ability to notify developers of changes.

Emails are important work products. Many OSS projects provide several mailing lists for different purposes. The most common mailing lists are announcement, support, developer, issue, and commit mailing lists. The developer mailing list is used for discussion related to changes to the source code. The issue mailing list is used to record changes in the project's issue tracking system. The commit mailing list is used to record changes made to the code repository. Both issue and commit mailing lists are updated automatically by the corresponding systems and are read-only. Many OSS projects conduct patch review through the developer mailing lists. A patch can be sent as an email and the community can comment on it by replying with feedback. The traffic of these mailing lists varies between OSS projects. The developer mailing list is one of the primary data sources for collecting data on a patch review process in various studies.

B. *Process Roles*

There are several roles individuals can have in the patch review process. Although studies differ in role nomenclature, we identify them using the terms issue reporter, patch contributor, peer developer, committer, and reviewer. *Issue reporters* are project participants who report problems to the communities through available channels such as a developer mailing list or an issue tracking system. Issue reporters may or may not have the skills or time to provide solutions to problems, but are impacted by problems enough to file reports. *Patch contributors* are project participants who try to address issues by contributing solutions. Patch contributors often do not have permission to modify the project source code repository directly, working instead with committers [12, 17]. *Peer developers* provide feedback on the work of the contributor [19]. Usually, one of them is a *committer*. The committer is an experienced developer who has permission to modify the project source code directly [12]. In order for a patch to be applied to a project's codebase, it normally has to be approved by *reviewers*. The reviewers are highly experienced developers in the area of code that the patches modify.

C. *Activities in the Patch Contribution Process*

We identified 6 common activities across multiple open source projects. Fig 2. shows activities and roles in the patch contribution process. These activities include 1) Patch Creation, 2) Patch Publication, 3) Patch Discovery, 4) Patch Review, and 5) Patch Application.

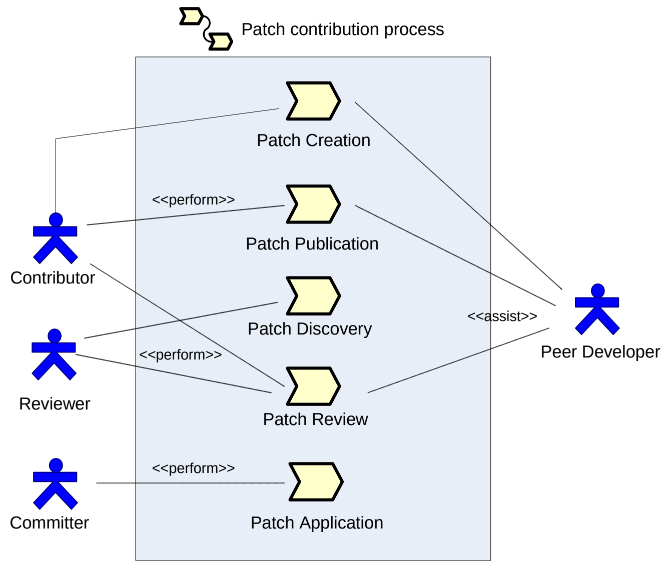

Figure 2. Patch contribution use cases

1) *Patch Creation:*

[2] For example Bugzilla (http://www.bugzilla.org/) or Trac (http://trac.edgewall.org/)

[3] For example Gerrit (http://review.source.android.com/) or Reviewboard (http://www.reviewboard.org/)

The patch contribution process starts when a contributor is ready to share the changes with the community. We assume that the changes the contributor intends to make have not been provided by other contributors. The contributor is responsible for creating patches. The goal of this activity is to create a patch that is complete, correct, and conforms to project standards. Common contributor tasks during patch creation are 1) modifying the code and 2) preparing the patch.

a) Modifying the Code

This task is the most commonly documented across all ten projects. Before creating patches, the contributor is expected to test the modified code, check the code against the project coding standard and have the right to share the code based on the license of the contributed OSS project. When creating patches, the contributor should follow the guidelines provided by an open source project. To increase the likelihood of the patch being accepted, it is important that the patch format conform to standards of the contributed project. Once a patch is created, the contributor has to make sure that it works. To do so, the contributor must try to apply the patch against the latest development version and test the code.

We found several common recommendations among the projects.

1. Every project requires that the patch must conform to the project coding standard. The coding standard is a document that describes the preferred coding styles used by an OSS project. While each project may use different tools, there are two prevalent approaches to creating patches. For a simple patch, contributors may use the diff program to create the patch. For patches that change multiple files, contributors should use the command provided by a revision control client such as CVS, SVN or Git.

2. The required patch format is the same across almost all projects, namely a “unified” format. The sole exception studied is PostgreSQL, who recommends a “context” format but allows a unified format. Patches can be hard to read without enough background information. The Mozilla project recommends using 8 lines of context for each patch hunk so reviewers can understand the patch without opening the source file.

3. Patches should be created against the latest development version. This is to ensure that the bug exists in the latest version of the source code. The change can then be back-ported to older versions if needed. Linux, Apache, Mozilla, KDE, Drupal, and Android explicitly recommend doing this. Drupal recommends attaching a patch for each version if needed.

4. It is common that patches need to be revised. Thus, a patch should be uniquely named and should include version number as part of the patch name.

Some recommendations can also be generalized to other projects. Smaller patches are likely to be accepted since it is easier and take less time for a reviewer to review. The Linux project recommends that a contributor separates patches such that each patch represents a single logical change, rather than including multiple changes within the same patch. This makes patches independent and potentially smaller, thus makes it easier for the reviewer to review them. On the other hand, PostgreSQL only recommends the smallest change that can be release without other patches.

b) Preparing the Patch

Once the patch is ready, the contributor then has to prepare information to support the patch based on the guidelines given by the contributed project. In general, patches need at least to be accompanied by a one-line description of what the patch does and a short paragraph explaining why the patch should be added to the contributed project source code. If the patch is a fix for a bug tracked by the bug tracking system, then the bug identification number should also be included.

In addition to the required information, extra information can help reviewers better evaluate the patch and thus increase the likelihood of its being accepted. For example, PostgreSQL requires additional information such as the project name, the status of the patch (work-in-progress or complete), the branch where the patch should be applied, test results, documentation on how to use the new feature, and description of how the change affects performance. For the revised patch, Linux recommend contributors provide enough background information about the changes from the previous patch to help reviewers remember the detailed discussion that should have occurred prior to the revision.

2) Patch Publication

When a contributor wants to share their patch, the patch has to be published to the channels provided by the contributed project. This activity is performed by the contributor but may involve interaction with people from the developer community in order to get required information. The goal of this activity is to publish the patch where reviewers can access it. The tasks that the contributor has to perform are to determine a publication channel, and to publish the patch.

a) Determining a Publication Channel

We identified three different publication channels for patches: project mailing lists, issue/bug tracking systems, and patch tracking systems. Most projects suggest one channel over another, but some projects allow more than one publishing channel. The Linux Kernel project is the only project that uses a mailing list as the sole publication channel. Apache, Drupal, Mozilla, OpenOffice.org, and Ubuntu require contributors to submit a patch to an issue tracking system. X.org uses a mailing list in conjunction with an issue tracking system. In KDE, where each sub-project operates semi-independently, the publishing channels are decided by sub-project. Thus, in KDE contributors have to first locate the appropriate publishing channel. Android and some of the KDE projects requires contributors to use the project patch tracking system. PostgreSQL is an interesting case. They use a mailing list as a publishing channel but they use a patch tracking tool to complement the mailing list. Patches are submitted to the mailing list but contributors are recommended to add the patch to the patch tracking system. This system reportedly provides better visibility of patch-related activity to both reviewers and contributors.

Regardless of the publishing channel, the contributor needs to gather the patch and its documents, contact information for the potential reviewers, the destination where the patch will be applied, and the project name (in case the channel supports several projects). Getting contact information for potential reviewers is not an easy task, particularly for new contributors. There is often no specific place to get this information. Contributors may have to use various tools to get information.

b) Publishing a Patch

Having selected a channel, the contributor has to package the patch based on project guidelines. Then the contributor can publish the patch at the selected channel and wait for feedback. The steps on how to publish the patch depend on the tools used for publication.

An issue tracking system is the most common publishing channel. However, issue tracking tools are not common across every project. OpenOffice.org, Eclipse, X.org, and Mozilla use Bugzilla or a variant. Drupal and KDE have their own tools. To publish a patch there, user can either attach a patch to an existing issue report or create a new issue report. Every project recommends that contributors should try to find an existing issue report that covers the patch before creating a new one. Adding a patch to an existing issue will make it visible to others who follow the activity of the issue report. Each project has different ways to attach patches to issue reports. Apache uses a specific keyword for "issue with patch". Drupal and OpenOffice.org have a specific issue status for patches. Eclipse Mylyn, recommends that users add "[Patch]" to the bug summary.

Fig. 3 represents common steps for publishing patches through an issue tracking system. The contributor needs to have an account to access the issue tracking system. Once the contributor logs into the system, she/he needs to identify an issue that addresses the patch. This can be done using the search feature of the issue tracking system. OSS projects guidelines often recommend spending a few minutes searching. If no relevant issue is found, then the contributor has to create a new issue for the patch, describe the issue and identify the reviewers. To publish the patch to the issue, the contributor can upload the patch as an attachment to the issue and update the issue information to show that the issue has a patch to be reviewed.

Email is the oldest publishing channel. However, many OSS projects have migrated away from it. To publish a patch using email is quite simple. Linux and X.org require contributors to put content of the patch in the body of the email. However, PostgreSQL recommends contributors attach a patch file to the email. A contributor may be expected to specify the email addresses of potential reviewers. The potential reviewers are developers who are familiar with the code affected by the patch. Contributors are recommended to send patches to the developer mailing list in order to make them visible to other developers.

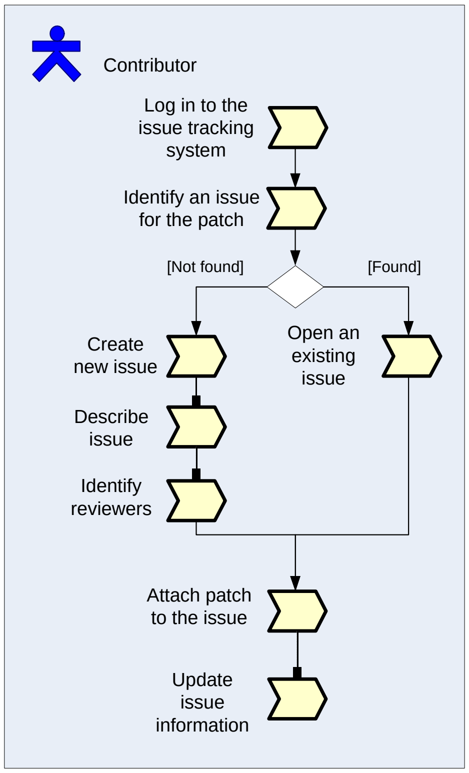

Figure 3. 'Publishing Patch' activity diagram

A patch tracking system is the most recent publishing channel, though few projects have adopted it. The patch tracking system is a web-based application that is designed to support patch review process. It provide an easy way for contributors and reviewers to publish, track, search, and review patches. The KOrganizer and Android projects use a patch tracking system to facilitate patch review and communication. Linux now provides a simple patch tracking system that can report the status of patches. PostgreSQL has a patch tracking system that is integrated with an email archive. Users can use it to check patch status and give feedback. Android has the most integrated system to support the patch publication. It integrates patch submission with its revision control system. Every patch must appear in the patch tracking system before it can be committed in to the main source repository.

3) Patch Discovery:

This activity involves the contributor, reviewers, peer contributors, and committers. The fact that many patches are lost or ignored makes this activity especially important. It's important for both contributors and OSS projects to be able to track the contributed patches and provide a better way to let potential reviewers know about a new patch. There are many ways for reviewers to discover patches; the publishing channel plays an important role here.

If a patch is published to a mailing list or emailed directly to the reviewer, reviewers can discover the patch by checking their email. However, there is no way for a contributor to know whether anyone has discovered the patch unless someone replies to the email. It is not likely that reviewers will reply to the patch to acknowledge that they have glanced at it. The contributor is likely to have to wait until reviewers have time to read the patch and give feedback. Unfortunately this can take a long time.

Using a mailing list as a sole publishing channel requires reviewers to manage email that contains patches. Email by itself is not efficient for managing a collaborative task because it requires reviewers to be good at task management [2, 31], [34]. As a result, patches may be forgotten, or the amount of time that it takes reviewers to keep and find patches may cause reviewers to give up.

If a patch is published to an issue tracking or patch tracking system, reviewers can discover the patch by looking at issue reports assigned to them. For systems that have email capability, reviewers can be notified by email when reports are modified. Thus, reviewers can discover patches by checking their email, although that leads to many of the problems described above.

The general recommendation for making patches easier to discover is to provide a tool to display a list of all active patches. Projects can use a patch tracking system or develop their own. The Drupal project provides a link to a list of active patches from their main web page. This makes it easy for reviewers to find patches. The PostgreSQL uses a patch tracking tool, CommitFest[4], and allocates a specific period of time for patch review during each development cycle.

4) Patch Review:

This activity involves the contributor, reviewers, peer contributors, and committers. Patch review has been extensively studied by researchers because of its importance [1, 19, 25]. Interestingly, it is the activity that has the least documentation on most OSS project websites. The goal of patch review is to decide whether or not the patch should be applied to the project source code.
Fig. 4 explains how patch review is performed. We identified three major tasks: verifying the patch, refining the patch and resolving the patch.

a) Verifying a Patch

Anyone who can understand a patch may verify its quality and suggest quality improvements. The focus of the review is to make sure that the patch is qualified to be included in the project master codebase. Reviewers review the patch based on several technical criteria such as quality, security, maintainability, integration, testing and licensing. Several questions can guide a reviewer. Does this change follow the project standard? Does this change introduce flaws that will cause problems in the future? Is this change a good way to solve the problem? Does this change introduce any security or instability risks? Does the code follow the code licensing rules? Does the changed code work properly with other modules?

[4] CommitFest tool https://commitfest.postgresql.org/

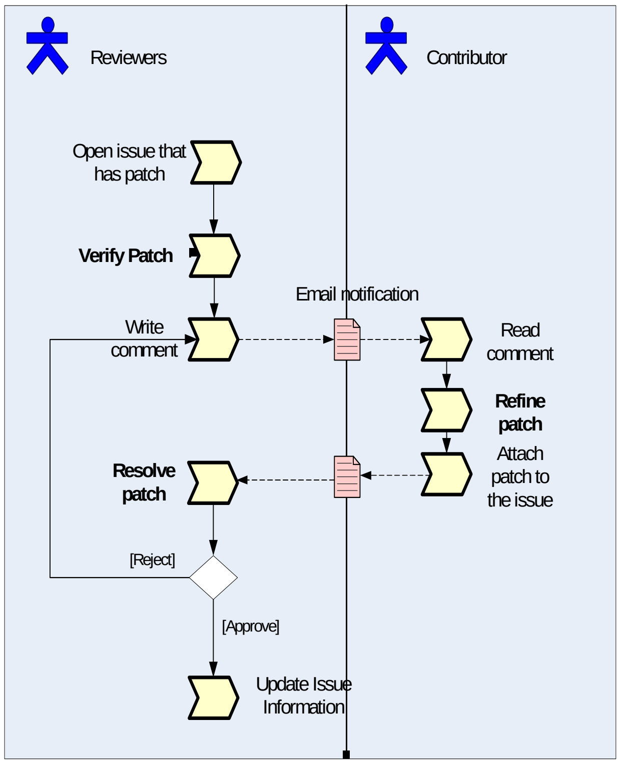

Figure 4. *'Patch Review' activity diagram*

The most common verification technique used in OSS projects is asynchronous code review. Code review is performed by an experienced developer who has technical expertise or domain expertise in the part of the system affected by the patch. The reviewer will read the patch and its supporting information to verify the quality of the patch and suggest improvements.

When a patch is sent by email to the mailing list, the reviewer can review the patch by reading the email. Providing a high-quality patch is one way to gain respect from reviewers. An email that contains enough information for reviewers lets them focus on the review rather than information gathering. This increases the possibility of being reviewed, since reviewers tend to ignore an incomplete patch. A good patch makes the process more efficient and takes less time to finish. Reviewers can give feedback on the patch by email either directly to the contributor or to the community. Putting a patch in the email body makes it easier for a reviewer give feedback about a specific part of the patch.

Experienced reviewers can understand the patch by simple inspection, but this can be hard for the beginners. Recently, web-based code review tools have been developed to ease understanding of code changes. The most common capabilities include the ability to compare the original source and the proposed changes, and the ability to comment on the code inline while maintaining the full code view. As of yet, not many of the studied OSS projects have adopted the new tools. Only KDE-PIM and Android use these tools to support the code review task. Mozilla added side-by-side code comparison to their bug tracking system.

b) Refining a Patch

A contributor is expected to address feedback received from reviewers. Patch refinement is an iterative process. The contributor may have to modify the patch many times before it can be accepted. In order to keep the reviewer well-informed of the history of patch modifications, the contributor is recommended to write a summary of new changes when resubmitting a new version of the patch. The goal of patch refinement is to improve the patch to an acceptable level before it is included in the main source code. The success of this task depends largely on the ability of the contributor to communicate and work together with reviewers in order to reach resolution. Thus, it provides a great opportunity for the contributors to demonstrate their technical skills as well as social skills to the community. As contributors demonstrate these skills they will gain trust from the community and increase their reputation. The larger open source community is a technical meritocracy that tends to pay more attention to those who successfully contribute to projects [21].

There can of course be conflicts between the contributor and reviewers. The openness of the review process usually makes it possible to resolve the conflicts through discussion. Community members can work together based on the recorded information to resolve conflicts. In some case, conflicts may need to be resolved with the help of people who are highly respected by the community.

c) Resolving a Patch

The resolution of a patch is its final disposition; normally a patch is either accepted or permanently rejected. Different projects have different approaches to resolving patches. In most cases, patch resolution is done by the core developers responsible for the relevant portions of the source code. These developers make the final decision as to whether or not each patch will be committed to the project codebase.

We observe several resolution approaches in the studied projects. Linux uses a staging approval approach. A patch has to go through approval multiple times, from subsystem maintainers and then up the project hierarchy. The Apache project uses a voting approach. A patch needs three positive approvals and zero negative vetoes in order to be accepted. The Mozilla project uses a two-reviewer approach. Most patches have to be approved by every owner of modules affected by the patch, and also by a “super-reviewer” who focuses on high-level impacts.

Although rejected patches can be resubmitted, some patches may be rejected regardless of the number of resubmissions--for example, a patch that fixes a problem that has already been fixed by other patches or a patch that adds a new feature that is not acceptable as part of the project vision.

5) Patch Application:

Patches can be applied to the project codebase by anyone who is a committer for the module modified by the patch. In most cases, one of the people who participate in the patch review should be a committer, and that person will apply the patch. Sometimes reviewers can also apply the patch. Tools are available to apply well-formatted patches automatically, but most OSS projects apply patches semi-manually.

The accepted patch has to be collected by a committer. This can be done in several ways. Ideally it should be done by a committer who gave feedback on the patch and is thus currently in position of its final version. However, if the accepted patch has somehow left the workflow, the contributor can contact the committer so that the patch can be applied. Depending on the project, the committer may commit patches one at a time or use a tool to help apply multiple patches at once. The Mozilla project has a specific keyword for an accepted patch to make it easier for committers to search from the bug tracking system. PostgreSQL’s CommitFest provides a list of patches that are ready to be committed[5]. Committers can go through the list and apply patches that are ready.

To apply a patch, the committer needs at minimum a patch file and a source code management tool. The committer has to apply the patch to the local working source code, then use the source code management tool to apply changes to the project codebase. However, before doing so, the committer may need to perform final basic tests to make sure that the patch does not cause problems. In general, patches that reach this stage should be free from trivial problems. In the Android project, patch application is done automatically by the patch tracking system. Patches are applied to the main source code automatically when reviewers approve them.

V. CONCLUSION

Open source projects are developed through a process of patch contribution. Proper design and management of this patch contribution process is a key contributor to project sustainability. Existing patch contribution processes vary from project to project, but there are a number of commonalities that comprise likely best practices for patch contribution.

By studying ten disparate successful open source projects, we have identified a collection of key practices for patch contribution. Together with some general observations about patch contribution, these form the basis of our architecture for a managed patch contribution process. We identify five basic stages of the patch contribution process: patch creation, discovery, publication, review and

[5] For a list of patches in the CommitFest see https://commitfest.postgresql.org/

application. Each of these stages is itself decomposed into one or more steps.

Our study of open source projects generally supports the idea that a managed patch contribution process improves project sustainability. Studied projects report that improved patch contribution practices have resulted in project improvement. Since open source is created by patch contribution, inefficiencies and failures in patch contribution would be expected to impair open source creation, so this is not a surprising finding; other researchers have also reported this to be the case [5, 8].

However, attention both by open source project leaders and in the research community has typically been focused on one or two particularly troublesome steps of the process. We believe that a view of patch contribution that treats the entire patch lifecycle can realize the gains implied by these lower-level individual improvements across a wide array of open source projects. In particular, adoption of better patch discovery practices looks to have impact across a large number of projects.

Evidence of sustainability of open source projects is a key driver for industry adoption of open source. This is especially true when industry itself seeks to be a primary contributor. For open source projects to be sustainable patch contributions must be managed. We believe that adopting our suggested patch contribution process will provide quantifiable gains in sustainability.

REFERENCES

[1] Asundi, J. and Jayant, R., “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, p. 166c, 2007.

[2] Bellotti, V., Ducheneaut, N., Howard, M. and Smith, I., “Taking email to task: the design and evaluation of a task management centered email tool,” *Proceedings of the SIGCHI conference on Human factors in computing systems*, Ft. Lauderdale, Florida, USA: ACM, pp. 345-352, 2003.

[3] Bergquist, M. and Ljungberg, J., “The power of gifts: organizing social relationships in open source communities,” *Information Systems Journal*, vol. 11, pp. 320, 305, 2001.

[4] Bird, C., Gourley, A. and Devanbu, P., “Detecting Patch Submission and Acceptance in OSS Projects,” *Proceedings of the Fourth International Workshop on Mining Software Repositories*, IEEE Computer Society, p. 26, 2007.

[5] Bird, C., Gourley, A., Devanbu, P., Swaminathan, A. and Hsu, G., “Open Borders? Immigration in Open Source Projects,” *Proceedings of the Fourth International Workshop on Mining Software Repositories*, IEEE Computer Society, p. 6, 2007.

[6] Bollinger, T., "Use of Free and Open-Source Software (FOSS) in the U.S. Department of Defense", *Technical Report# MP 02 W0000101,* The MITRE Corporation, 2003.

[7] Crowston, K., Li, Q., Wei, K., Eseryel, U.Y. and Howison, J., “Self-organization of teams for free/libre open source software development,” *Inf. Softw. Technol.*, vol. 49, pp. 564-575, 2007.

[8] Dahlander, L. and Magnusson, M.G., “Relationships between open source software companies and communities: Observations from Nordic firms,” *Research Policy*, vol. 34, pp. 481-493, May. 2005.

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

[10] DiBona, C., Ockman, S. and Stone, M. eds., *Open Sources: Voices from the Open Source Revolution*, O'Reilly \&amp; Associates, Inc., 1999.

[11] Hars, A. and Ou, S., “Working for Free? Motivations for Participating in Open-Source Projects,” *Int. J. Electron. Commerce*, vol. 6, pp. 25-39, 2002.

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

[13] von Krogh, G., Spaeth, S. and Lakhani, K.R., “Community, joining, and specialization in open source software innovation: a case study,” *Research Policy*, vol. 32, pp. 1217-1241, Jul. 2003.

[14] Lakhani, K. and Wolf, R.G., “Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects,” *SSRN Electronic Journal*, 2003.

[15] Lakhani, K.R. and Hippel, E.V., “How open source software works: “free” user-to-user assistance,” *Research Policy*, vol. 32, pp. 923-943, Jun. 2003.

[16] Lonchamp, J., “Open Source Software Development Process Modeling,” *Software Process Modeling*, 2005.

[17] Mockus, A., Fielding, R.T. and Herbsleb, J., “A case study of open source software development: the Apache server,” *ICSE '00: Proceedings of the 22nd international conference on Software engineering*, New York, NY, USA: ACM Press, p. 263—272, 2000.

[18] Mockus, A., Fielding, R.T. and Herbsleb, J.D., “Two case studies of open source software development: Apache and Mozilla,” *ACM Trans. Softw. Eng. Methodol.*, vol. 11, pp. 309-346, 2002.

[19] Nurolahzade, M., Nasehi, S.M., Khandkar, S.H. and Rawal, S., “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, pp. 9-18, 2009.

[20] Perry, D.E., Porter, A., Wade, M.W., Votta, L.G. and Perpich, J., “Reducing inspection interval in large-scale software development,” *IEEE Trans. Softw. Eng.*, vol. 28, pp. 695-705, 2002.

[21] Raymond, E.S., *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*, O'Reilly & Associates, Inc., 2001.

[22] Reddy, B. and Evans, David S., “Government Preferences for Promoting Open-Source Software: A Solution in Search of a Problem,” *Social Science Research Network*, working paper, May 2002.

[23] Reis, C.R. and Fortes, R.P.D.M., “An Overview of the Software Engineering Process and Tools in the Mozilla Project,” *Workshop on OSS Development*, Newcastle upon Tyne, UK: 2002, pp. 162-182.

[24] Rigby, P. and German, D., "A preliminary examination of code review processes in open source projects", *Technical Report# DCS-305-IR*, University of Victoria, 2006.

[25] Rigby, P.C., German, D.M. and Storey, M., “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.

[26] Ruffin, M. and Ebert, C., “Using Open Source Software in Product Development: A Primer,” *IEEE Software*, vol. 21, pp. 82-86, 2004.

[27] Scacchi, W., “Understanding the requirements for developing open source software systems,” *IEE Proceedings - Software*, vol. 149, pp. 24-39, 2002.

[28] Schmidt, K.M. and Schnitzer, M., “Public Subsidies for Open Source - Some Economic Policy Issues of the Software Market,” *Harvard Journal of Law & Technology*, vol. 16, p. 473, 2002.

[29] Stamelos, I., Angelis, L., Oikonomou, A. and Bleris, G.L., “Code quality analysis in open source software development,” *INFORMATION SYSTEMS JOURNAL*, vol. 12, pp. 43--60, 2002.

[30] Stewart, K.J., Ammeter, A.P. and Maruping, L.M., “A Preliminary Analysis of the Influences of Licensing and Organizational Sponsorship on Success in Open Source Projects,” *Hawaii International Conference on System Sciences*, Los Alamitos, CA, USA: IEEE Computer Society, p. 197c, 2005.

[31] Venolia, G.D., Dabbish, L., Cadiz, J. and Gupta, A., "Supporting Email Workflow," *Microsoft Research Tech. Report MSR-TR-*

2001-88, Microsoft Corporation, 2001.

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

[33] West, J., "How open is open enough?: Melding proprietary and open source platform strategies," *Research Policy*, vol. 32, pp. 1259-1285, Jul. 2003.

[34] Whittaker, S., "Supporting collaborative task management in e-mail," *Hum.-Comput. Interact.*, vol. 20, pp. 49-88, 2005.