Small Patches Get In!

Peter Weißgerber
Computer Science
University of Trier
54286 Trier, Germany
weissger@uni-trier.de

Daniel Neu
Computer Science
University of Trier
54286 Trier, Germany
danielneu81@googlemail.com

Stephan Diehl
Computer Science
University of Trier
54286 Trier, Germany
diehl@uni-trier.de

ABSTRACT

While there is a considerable amount of research on analyzing the change information stored in software repositories, only few researcher have looked at software changes contained in email archives in form of patches. In this paper we look at the email archives of two open source projects and answer questions like the following: How many emails contain patches? How long does it take for a patch to be accepted? Does the size of the patch influence its chances to be accepted or the duration until it gets accepted? Obviously, the answers to these questions can be helpful for the authors of patches, in particular because some of the answers are surprising.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*documentation, restructuring, reverse engineering, and reengineering, version control*; D.2.9 [**Software Engineering**]: Management—*programming teams*

General Terms

Documentation, Experimentation, Measurement

Keywords

Email archives, Patches, Patch acceptance, Case study

1. INTRODUCTION

For virtually every open source project there exists a developer mailing list. For many open source projects this mailing list can also be read by everyone who is interested using a web front-end. One important function of such a mailing list is that people can submit *patches*. Patches are change sets that can be applied to the software using a specific tool: the `patch` tool. Patches may, for example, introduce new features, fix bugs, translate strings to a different language, or re-structure parts of the software (by applying refactorings).

So far, change information in software repositories like CVS has been leveraged by tools for defect prediction, recommender systems and the like. The change information in mailing lists could be used in a similar way or to improve these tools. Unfortunately, not all patches submitted actually make it into the real software and thus may lead to wrong predictions or suggestions. Only few studies [1, 2, 4] exist that address the submission and acceptance of patches. In this paper we want to examine by means of a case study, which data we can obtain by combining patch extraction from mailing list archives with detection of patch acceptance in CVS repositories, and what we can learn from these data.

1.1 Patch submitters and CVS committers

In this paper we distinguish two not necessarily disjoint groups of developers: *submitters* and *committers*. Submitters are people who send patches to the developer mailing list of a project, while committers check-in new revisions of files to the CVS respectively Subversion repository of the project.

Usually submitters have at least read-access to the source code of the project[1]. Thus, they can read and review the source code, and try to improve things or add new functionality. There are two typical reasons for submitters to send their patches to the mailing list: either they want to discuss their changes with others, or they want that one of the committers changes the code in the software repository accordingly by applying the patch and committing the resulting file revision. Note, that also committers sometimes send patches to the mailing list for discussion and become submitters this way.

One part of our study focuses on the submitters and committers. We explore which group is larger, and if there is a group of core submitters. Furthermore, we take a closer look on how these groups intersect.

1.2 Getting the patch accepted

In any case, a patch sent to the mailing list may be applied to the source code of the project and the generated new revision of the particular file committed to the repository at a later time. In this case, we say the patch has been *accepted*. It goes without saying that not all patches must be accepted.

Obviously, a patch submitter hopes that his or her patch

[1]As in open-source projects the source code is available for free, everyone who is interested can become a submitter quite easily.

is accepted, and that it is accepted within a preferably short amount of time. Thus, one focus of our case study is to examine how many percent of the submitted patches are accepted at all, and how long it takes until patches are accepted. We also address the question, if it is easier to get small patches accepted than large ones.

The paper is organized as follows: In Sections 2 and 3 we describe how we extract patches from the emails contained in mailing list archives and how we find out for each patch if it has been accepted. In these sections we use mainly techniques which have been presented by Christian Bird et al. in their last year's MSR paper [2]. Furthermore, we define exactly when we consider a patch as being accepted. The most important part of this paper is Section 4 which contains the case study on patch submission and acceptance. In Section 5 we refer to and discuss related work. Possible threats to validity are discussed in Section 6. Finally we conclude in Section 7.

2. PATCH EXTRACTION

According to Bird et al.'s technique we implemented a parser that extracts patches from e-mail archives in MBOX format. MBOX is a standardized format (see RFC 4155) to store a set of emails, e.g. all emails sent to a particular address or mailing list. Thus, for many open-source projects it is possible to obtain a MBOX file containing the complete mailing list archive. An MBOX file for local mails is also easily obtainable, because some email clients, e.g. Mozilla Thunderbird, store their local email folders in MBOX format.

Patches can be in three different formats: standard diff, context diff, and unified diff. Standard diffs only describe which lines (identified by line numbers) are added or deleted, but they do not contain any context information. They even do not contain the name of the file to be changed. As this makes it nearly impossible to detect these patches in the CVS repository within an acceptable time frame and with an acceptable exactness, we omit standard diffs. In our opinion this is not a big limitation: because of the missing context such diffs are not very useful for developers and are only used very seldom[2].

As both the unified diff and the context diff have a well-defined header which contains the name of the patched file and the length of the diff, it is quite easy to extract patches of these types. The only problem, as noticed by Bird et al., is that sometimes mail clients add extra line breaks. As these line breaks are inserted before the mail is actually sent, they will also appear in the MBOX. However, it is easy to hand-tune the parser in a way that it works reasonably exact.

3. PATCH DETECTION

In Section 2 we have extracted patches out of emails. In the following we explain how we find out whether a patch has been accepted, i.e. applied by a developer to the sources of the project and committed to the CVS resp. Subversion repository.

[2]We manually inspected the MBOX files for FLAC and OpenAFS and found that only 3 and 24 patches resp. are in standard diff format and thus ignored, while 196 and 1628 patches resp. have been recognized by our approach.

As patches of type context diff and unified diff contain the name of the modified file in their header, the main idea is to look for the application of the patch in all revisions of that file that are checked-in at a later time than the patch has been sent to the mailing list (with respective to the timestamps of the revision resp. the email containing the particular patch).

In detail our algorithm to detect submitted patches in the CVS repository works as follows: We treat each patch as a tuple $(d_p, f_p, t_p, A_p, R_p)$ where d_p is the directory of the patched file as specified in the particular diff, f_p is the name of the patched file as specified in the particular diff, t_p is the time stamp of the patch[3] which is obtained from the time stamp of the email containing that patch, A_p is the set of lines added by the patch (for each line leading and ending white spaces are removed), and R_p is the set of lines removed by the patch (for each line leading and ending white spaces are removed). Note that in our definition each patch addresses exactly one file. Thus, if a submitter wants multiple files to be changed, the particular email will contain several patches.

Let the tuple $(d_r, f_r, v_r, t_r, A_r, R_r)$ be a file revision in the repository, where d_r is the directory, f_r is the filename, v_r is the revision number, t_r is the time stamp when the revision has been committed, A_r is the set of added lines with respect to the predecessor version, and R_r is the set of removed lines with respect to the predecessor version. Again, for each line in A_r and in R_r leading and ending white spaces are removed. If no predecessor version exists, A_r contains all lines present in the revision, and R_r is the empty set.

We consider a patch $(d_p, f_p, t_p, A_p, R_p)$ as being accepted, if a file revision $(d_r, f_r, t_r, v_r, A_r, R_r)$ exists in the repository, for which the following condition holds:

$$\begin{array}{ll} & d_p \sim_{\text{dir}} d_f \\ \wedge & f_p \sim_{\text{file}} f_r \\ \wedge & t_p \leq t_r \\ \wedge & A_p \subseteq A_r \\ \wedge & R_p \subseteq R_r \end{array}$$

Note that our formalization ignores both the line numbers of the added respectively removed lines, and the order of them. Furthermore, we also allow that the lines added by the patch and the lines deleted by the patch are only subsets of the lines added and deleted resp. of the new file revision. The reason for this it that we want to detect a patch application even if other changes have been performed within the same file or if the patch has been restructured to some extent, e.g. by adding comments between the added lines, or by changing the order of some lines.

One problem of detecting whether a patch has been accepted, is that the directory information of patches is often incomplete or even one directory in the path has been renamed. Analogously patch submitters tend to rename the patched file (e.g. by adding suffixes like ".new", "-patched", ..., or by renaming the extension itself). Thus, we do not require the directories and filenames of patch and revision to be exactly equal, but only to be *similar*.

In our heuristics $f_1 \sim_{\text{file}} f_2$ holds, iff f_1 and f_2 are equal,

[3]time stamps are treated as natural numbers representing the elapsed seconds since 1970-01-01 00:00:00.

or f_1 starts with f_2 but has a suffix, or f_1 and f_2 are equal except of different file extensions.

For directories $d_1 \sim_{\text{dir}} d_2$ is true, iff d_1 and d_2 are equal, or d_2 ends with d_1, or d_2 is equal to or ends with d_1 if only one part of d_1 is renamed ("one part" means one subdirectory in the directory path given by d_1).

4. CASE STUDY

We used the technique described above to perform a case study on two open source projects (see also Table 1):

FLAC is a tool for lossless compression of audio files. For FLAC we looked at the email archives of the development mailing list *flac-dev*[4] starting from January 2001 until the 18th July 2007. As we want to detect submitted patches in the CVS repository also for patches that have been submitted near the end of this time window, the observed time window for the CVS repository has to end later. We chose a look-ahead of 3 months, i.e. we examined the FLAC CVS repository within the time frame between January 2001 and the 18th October 2007.

OpenAFS is an open-source implementation of the network file system AFS. For OpenAFS we examined the archives of the *openafs-devel* mailing list[5] during the time period starting from the 5th November 2000 until the 11th June 2007. Again, our observation time period of the CVS repository is three months longer: it reaches to the 11th September 2007.

We selected these projects because for both the MBOX data can be downloaded directly and the CVS repositories can be accessed using a guest account. Unfortunately, for many other projects the mail data is only available via a custom web interface. We tried to parse the HTML code that is produced by SourceForge for their projects when accessing the mail archives. However, it turned out that parsing this generated HTML code is very tedious and often attachments—which may contain patches—cannot be accessed correctly. Thus, we were not able to obtain a reliable parser within an acceptable time frame (given that the time between the call for papers and the submission deadline was very short).

In this study we want to answer the following questions for the two selected projects:

- How many of the mails sent to the developer mailing list contain patches?
- How many files of the project are patched and which are patched most often?
- How often are accepted patches applied?
- What is the chance that a patch gets accepted? Is it influenced by the size of the patch?
- How long does it take until a patch is accepted? Is it influenced by the size of the patch?

[4]available at `http://lists.xiph.org/pipermail/flac-dev/`
[5]available at `https://lists.openafs.org/pipermail/openafs-devel/`

- How many people submit patches and how many people commit changes and patches to the repository. Do the sets of submitters and committers intersect? Is there a "core group" of submitters?

4.1 Basic Data

First of all, we want to find out how many mails sent to the mailing list contain patches at all. Furthermore, we take a closer look on which files are patched. One might think that the patches are equally distributed to all files of the project, but maybe in reality some files are patched more often than others. Additionally, we find out how often one patch is applied.

4.1.1 How many mails contain patches?

Table 2 shows for FLAC and OpenAFS how many mails have been sent to the developer mailing list, how many mails contained at least one patch, the ratio of mails with at least one patch, the total number of detected patches, and the average number of patches for mails with patches.

For FLAC in average every 26th–27th mail contains patches. Although only 82 emails contain patches, the total number of patches is 196. Thus, sometimes one mail contains multiple patches. We looked at the maximum number of patches in one mail and found it to be 32. However, most times when a mail contains patches, it only contains few ones: the average number of patches per mail (only for mails containing at least one patch) is 2.39.

For OpenAFS the patch ratio is slightly higher as for FLAC: here in average every 23rd mail contains at least one patch (i.e. 4.34% of all mails). The total number of patches for OpenAFS is 1628. For mails with patches the average number of patches is 3.63.

4.1.2 Patched Files

Project	total # files	# patched files	Ratio
FLAC	987	57	5.78%
OpenAFS	7823	460	5.88%

Table 3: Number of files and number of patched files for FLAC and OpenAFS.

Table 3 shows for both projects how many files exist in the particular CVS repository and how many files are patched at least once. Only accepted patches have been taken into account.

Although the total number of files is quite different for both projects (OpenAFS has about 8 times as much files than FLAC), the ratio of patched files is nearly the same: about 6% of all files are affected by at least one accepted patch.

Tables 4 and 5 show the most frequently patched files for both projects. In FLAC the mostly patched file has 10 revisions that contain patches. There are two files with 3 revisions with patches, 12 files with 2 revisions containing patches, 42 files with one revision containing patches, and 930 files that have not been patched at all. For the three files with the highest patch ratio, 50% resp. 33% of all revisions (except the initial revision) contain patches.

In OpenAFS the first rank in this list is occupied by the file src/afs/LINUX/osi_vnodeops.c. This file has been changed

Project	mailing list name	observed time period (mailing list)	observed time period (CVS repository)
FLAC	flac-dev	from 2001-01-01 to 2007-07-18	from 2001-01-01 to 2007-10-18
OpenAFS	openafs-devel	from 2000-11-05 to 2007-06-11	from 2000-11-05 to 2007-09-11

Table 1: Observed time periods for mailing list and CVS repository.

Project	# Mails	# MwP	Ratio	# Patches	minPpM	maxPpM	ANPMP
FLAC	2258	82	3.63%	196	1	37	2.39
OpenAFS	10317	448	4.34%	1628	1	142	3.63

Table 2: Basic Data (# MwP = number of mails with patches, minPpM / maxPpM = minimum / maximum number of patches in one mail, ANPMP = average number of patches for mails containing patches)

Filename	# chg	# cwp	Ratio
configure.in	151	10	6.62%
src/plugin_xmms/plugin.c	68	3	4.41%
src/libFLAC/Makefile.am	77	3	3.90%
12 files	-	2	-
42 files	-	1	-
930 files	-	0	-
src/flac/local_string_utils.c	2	1	50%
src/test_libFLAC/bitwriter.c	3	1	33.33%
src/share/utf8/utf8.c	6	2	33.33%

Table 4: Most frequently patched files (by total number of revisions with patches and by patch quote) for FLAC (chg = revisions of the file except the initial revision, cwp = revisions of the file containing at least one patch).

276 times. 17 of these 276 revisions—that are 6.16%—contain patches sent to the mailing list. However, only few files have a high number of revisions with patches. Anyhow, for this project there is even one file that has a patch ratio of 100%. However, this file has been changed only once. Overall, for both projects it seems that the files with high patch ratio are changed very seldom. The reason may be, that in general files are patched relatively seldom, and thus for files with a lower number of total changes, the ratio is automatically higher.

From Tables 3, 4 and 5 we can easily conclude that in both projects under view, some files are more likely to be patched than others.

4.1.3 How often are patches applied?

Naively one may think that a patch submitted to the mailing list has to be applied only once to the CVS repository of the software, if it is accepted. However, some situations can cause that an accepted patch has to be applied multiple times. Most notably it may be required to apply it to several development branches.

Both projects use branching mostly for maintenance, but also for development: The FLAC repository contains 9 branches at all. Looking only at the branch names, 7 of these can be clearly recognized as being maintenance branches. For OpenAFS we found 24 branches at all, from which 9 have a name that clearly identifies them as maintenance branches. The remaining branches are probably used for integrating new features or supporting new architectures. Let us take a look on how often accepted patches have been applied.

Filename	#chg	#cwp	Ratio
afs/LINUX/osi_vnodeops.c	276	17	6.16%
afs/VNOPS/afs_vnop_write.c	86	9	10.47%
WINNT/afsd/smb3.c	247	7	2.38%
afs/LINUX/osi_sleep.c	63	7	11.11%
libafs/MakefileProto.LINUX.in	93	7	7.53%
config/afs_sysnames.h	133	7	5.26%
4 files	-	6	-
3 files	-	5	-
17 files	-	4	-
36 files	-	3	-
89 files	-	2	-
305 files	-	1	-
7363 files	-	0	-
export/export5.exp	1	1	100%
aklog/linked_list.c	3	2	66.67%
afsd/rc.dkload.client.rs_aix	2	1	50%
afsd/rc.afs.rs_aix	2	1	50%
config/param.i386_fbsd_51.h	2	1	50%

Table 5: Most frequently patched files (by total number of revisions with patches and by patch quote) for OpenAFS (chg = revisions of the file except the initial revision, cwp = revisions of the file containing at least one patch).

Figure 1 shows for both projects how often accepted patches have been applied. In FLAC only 5 accepted patches out of 82 ones (i.e. about 6%) have been applied more than once, namely exactly twice. The remaining 77 accepted patches have been applied once. Thus, one can conclude that in FLAC accepted patches are normally applied only once. We find this a bit surprising because have expected that in more cases patches had been incorporated into the trunk as well as into at least one of the maintenance branches.

For OpenAFS the situation seems to be different: 57% (or in total 361 patches) have been applied once, but 43% have been applied multiple times. This means that nearly half of the accepted patches have been applied to more than one file revision! One patch has even been applied to seven different revisions—that is the maximum.

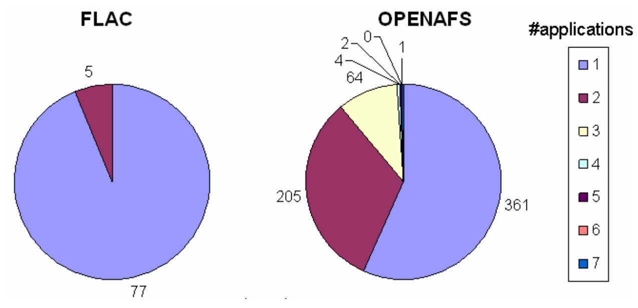

Figure 1: Number of applications per accepted patch

4.2 Patches and Patch Acceptance

In the following part of our case study we want to examine the acceptance of patches. For each person who submits patches or is interested in submitting patches, it is an interesting question how good the chances are to get the patch accepted. There might be several parameters that have an influence on the probability that a patch will get accepted. Intuitively, one might think that especially the size of the submitted patch has a high influence on the acceptance probability. Thus, we examine whether the acceptance ratio is different for different patch sizes.

4.2.1 How many patches get in?

Project	# submitted p.	# accepted p.	Ratio
FLAC	196	82	41.84%
OpenAFS	1628	637	39.13%

Table 6: Patch submission and acceptance in FLAC and OpenAFS.

Table 6 shows for both projects how many patches have been submitted, how many patches have been accepted, and the ratio of accepted patches to submitted patches.

Although for OpenAFS there are a lot more patch submissions than for FLAC (1628 vs. 196), the acceptance ratio is nearly equal for both projects. It is about 40%.

4.2.2 Does it depend on patch size?

As shown by Figure 2, slightly more than half of all submitted patches to FLAC change only one or two lines. The chances that such small patches get accepted are higher than average: 57% of the accepted patches are that small. This still holds if we look at patches which change at most 4 lines: While about 2/3 of all patches fall in this category, 78% of the accepted patches fall in it. The opposite holds for large patches (more than 15 lines of code changed): Despite this group includes 16% of all patches, only 3.6% of the accepted patches fall into this group.

For OpenAFS trends are the same as for FLAC: Figure 3 shows that about one third of the patches change at most two lines, but one third of the accepted patches fit in this category. Slightly more than one third of all submitted patches alter at most 4 lines, but 50% of the accepted patches fit in this category. 18% of all submitted patches to OpenAFS are large patches, i.e. change 25 or more lines. However, only 11% of the accepted patches are that large. In both projects patches that change two lines are the most popular ones.

Overall, the numbers allow us to conclude that small patches have a higher chance to be accepted than average, while large patches are less likely to be accepted.

4.3 How long does it take until a patch is accepted?

Another important question for a patch submitter is, how long it will take until the submitted patch is accepted and applied. Again, the duration may be influenced by several parameters and one might think that patch size has a big influence, in the sense that small patches are accepted quicker. We looked for FLAC and OpenAFS if this is true.

4.3.1 General Overview

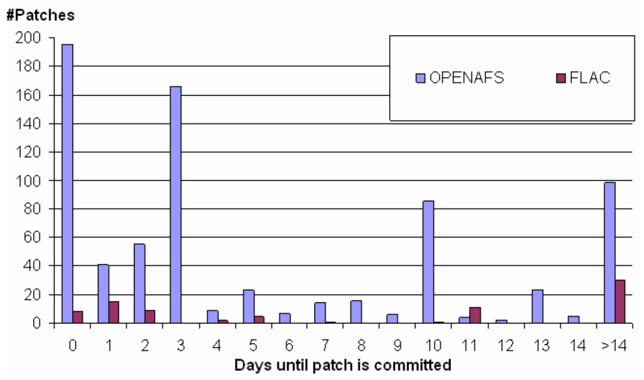

Figure 4: Number of days until a patch has been accepted

Figure 4 shows for FLAC and OpenAFS how many days it took for accepted patches until they have been committed[6]. We see that in both projects a lot of accepted patches have been committed to the repository within 3 days. The category with the highest number of patches is "at the same day" (0) for OpenAFS and "at the next day" (1) for FLAC. With rising number of days, the number of applied patches tends to fall. However, for both projects there are some peaks: For OpenAFS it catches the eye that especially after 3, after 10, and after 13 days many patches have been accepted. For FLAC we see an noticeable peak after 11 days.

Figure 5 shows the same values as a pie chart. While peaks are harder to observe, it is easier to recognize percentages: For FLAC one quarter of all accepted patches have been committed to the CVS repository on the same or the next day. Nearly half of all patches have been accepted within one week. For about one third of the accepted patches it took longer than two weeks until they have been committed.

For OpenAFS one quarter of all accepted patches have even been applied on the same day. After at most three days 61% of all accepted patches have been applied. Only 13% needed more than two weeks until being committed.

One can conclude that if patches are accepted, they normally are accepted quickly. For FLAC only one third and for OpenAFS even only 13% of the accepted patches needed longer than two weeks.

[6]For patches that have been committed multiple times, we took the first commit into account.

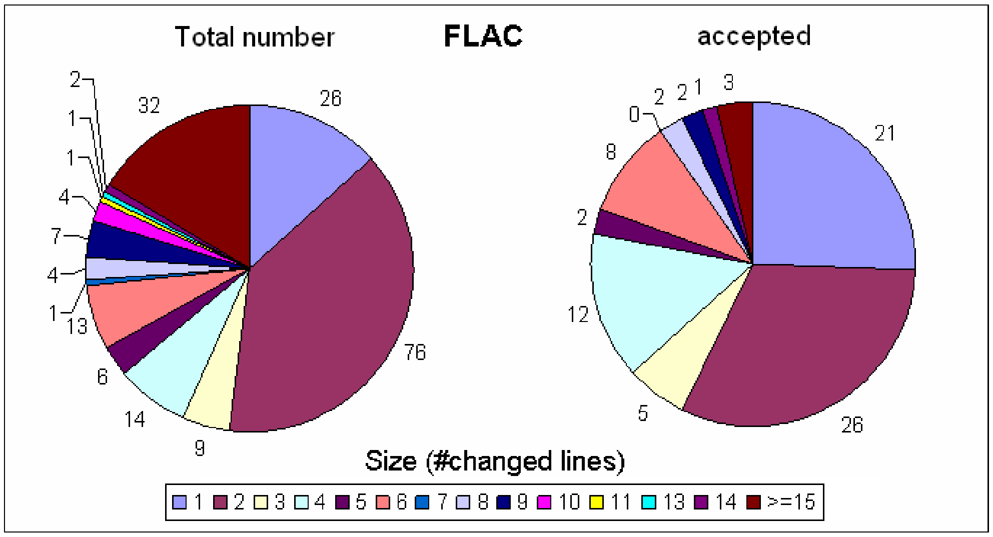

Figure 2: Patch size for the FLAC project

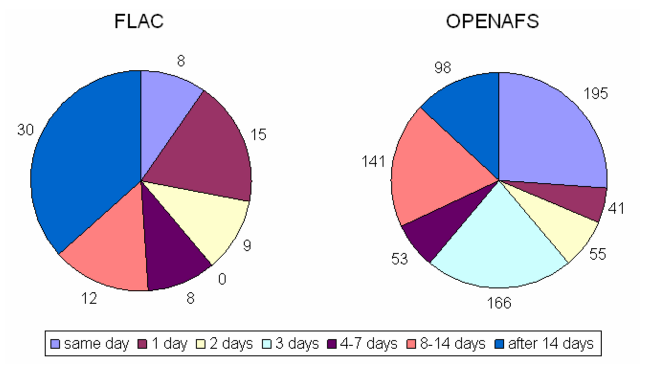

Figure 5: Number of days until a patch has been accepted

4.3.2 Does it depend on patch size?

To find out whether the duration until a patch is accepted depends on the size of the patch we look at Figures 6 and 7. These show for both projects how many days it took in average until a patch has been accepted (restricted only on accepted patches), broken down to the size of the patch.

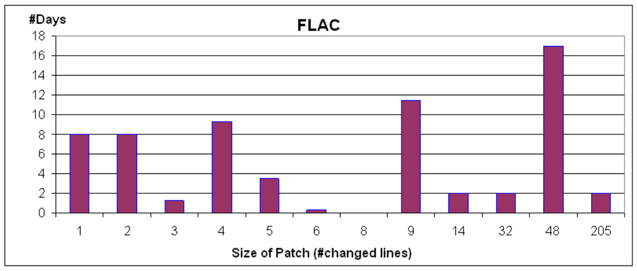

Figure 6: Average number of days until a patch is accepted, broken down by patch size (FLAC)

In FLAC the average is for all sizes around 1 to 10 days, while for OpenAFS it is mostly below ten days, excepting some outliers. However, overall one cannot determine that the patch size has a significant influence on the time until a patch is accepted. But note that, as shown in Section 4.2.2, at least for the examined projects it has an influence on the probability that a patch is accepted at all.

4.4 Submitters and Committers

While we have looked on the number and the acceptance of patches, as well as on the time it takes until patches are accepted, we have not yet regarded who submits the patches and who commits them, if they are accepted.

4.4.1 Submitters

First, we want to look at the patch submitters to find out if there are persons who submit patches very often and if there are submitters with an especially high success quote (in terms of accepted patches).

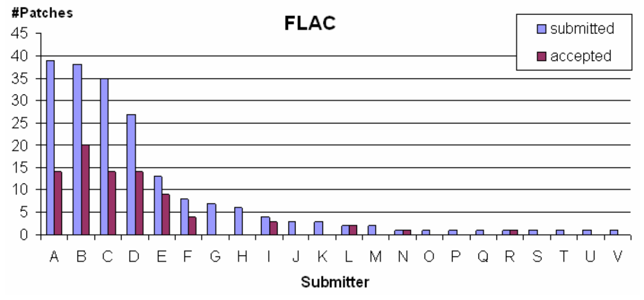

Figure 8: Patch submitters for the FLAC project and their success rate

Figure 8 shows for each patch submitter of the FLAC project how many patches he or she has submitted and how many of the submitted patches have been accepted. For privacy reasons we made the names of the submitters anonymous. The three most frequent submitters (Submitters A, B, C in Figure 8) have sent between 35 and 39 patches to

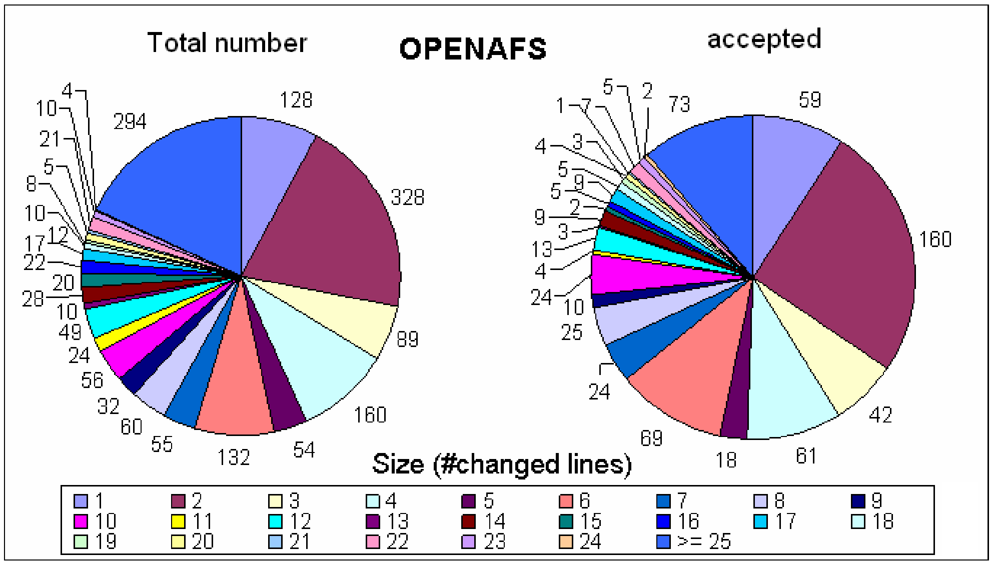

Figure 3: Patch size for the OpenAFS project

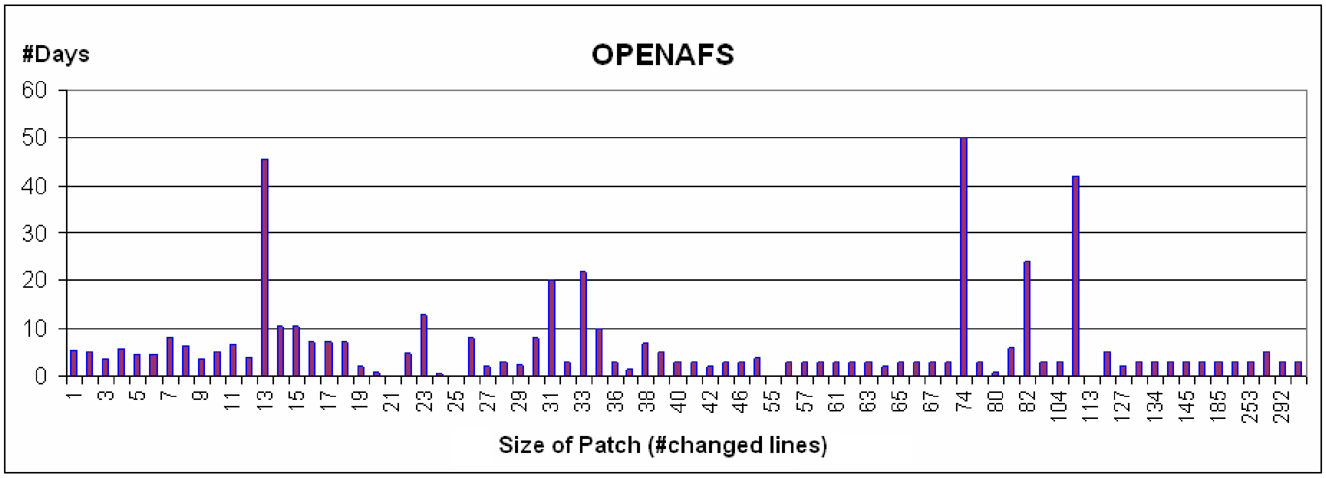

Figure 7: Average number of days until a patch is accepted, broken down by patch size (OpenAFS)

the mailing list. Submitter D follows with 27 patch submissions. Then there is a major jump: Submitter E has only submitted 13 patches. Thus, Submitters A to D seem to be the "hard core" of frequent submitters of FLAC.

The ratio of accepted patches does not vary that much between these submitters: ratios lie within a range of 36–53%. Thus, the success ratio of these most frequent patch submitters is near the average acceptance ratio of FLAC which is about 42% (see Table 6). There are three developers with a success ratio of 100%. However, they have only committed one single patch. From the submitters with more than 10 patches, Submitter E was most successful with a rate of 69% of accepted patches.

Figure 9 shows the same data for OpenAFS. One obvious fact is that OpenAFS has much more distinct submitters (112 submitters) than FLAC (22 submitters). It faces that Submitter A has submitted by far the highest number of patches (252 patches). Submitters B to E follow with 187, 170, 143, resp. 141 submissions. The next one, Submitter F only has submitted 63 patches. Thus, Submitter A can be seen as the main submitter and seems to build the team of core submitters together with B, C, and E (D has submitted many patches, but only very few have been accepted). The acceptance ratio for patches broken down to submitters is interesting for OpenAFS. Remember that the average acceptance ratio for OpenAFS is nearly 40% (Table 6). For Submitters A, B, C, and E the success quote is clearly higher than average (66% for A, 52–60% for B, C, E). However, the success ratio of D is only 10% and the success ratio of F even 0% although they have submitted many patches (C: 143, F: 63). When looking at the particular patches, it turns out that nearly all patches submitted by these two submitters were not for OpenAFS, but for the Linux kernel (in order to improve the compatibility of Linux with OpenAFS). This is interesting, because we expected that all patches sent to the mailing list of a project are change sets for exactly that

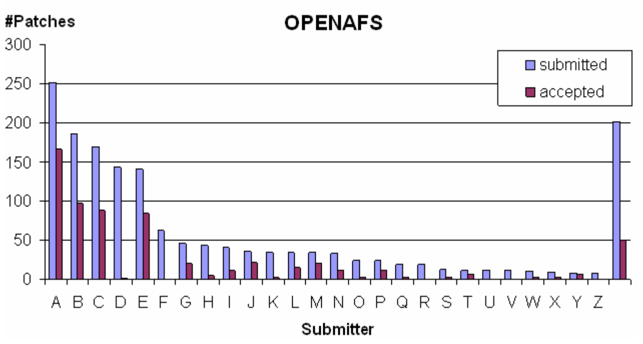

Figure 9: Patch submitters for the OpenAFS project and their success rate

project. However, this shows that some of the patches may also address other projects that are related in some way.

4.5 Committers

In the previous paragraphs we have examined who submits patches to the developer mailing lists. Next, we focus on the people who commit changes—among them applied patches—to the repositories: the committers. It is especially interesting how many committers a project has compared to the number of submitters, and if the check-ins of some committers contain more patches than of others. Furthermore we want to examine how often committers are also submitters, i.e. they send patches to the mailing list (probably to discuss their changes) although they would be able to commit these instantly.

Committer	IwS	# cfr	# pfr	ratio
1	P	10988	74	0.0067
2	n/a	78	0	0.0000
3	D	66	8	0.1212
4	n/a	5	0	0.0000

Table 7: FLAC committers (IwS = identical with submitter, cfr = number of changed file revisions, pfr = number of changed file revisions containing at least one patch).

Committer	IwS	# cfr	# pfr	ratio
1	S	36979	656	0.0177
2	yes	8457	25	0.0030
3	yes	3853	10	0.0026
4	n/a	995	1	0.0010
5	P	914	44	0.0481
6	M	549	10	0.0182
7	n/a	513	6	0.0117
8	yes	164	0	0.0000

Table 8: OpenAFS committers (IwS = identical with submitter, cfr = number of changed file revisions, pfr = number of changed file revisions containing at least one patch; yes in the IwS row means that the submitter is not one of the top-most 26).

Tables 7 and 8 show the committers of FLAC respectively OpenAFS together with the number of their changed file revisions (total and such that contain patches).

It strikes out that for both projects there are significantly less committers than submitters: FLAC has 22 submitters, but only 4 committers, giving a ratio of 0.18 between committers and submitters. Two of the four FLAC committers are also submitters. While Committer 3 is also a very frequent submitter (4th rank in the list of the most frequent submitters, see Figure 8), the by far most frequent committer only submits patches quite rarely (only 1 patch at all).

For OpenAFS the ratio of committers to submitters is even lower than for FLAC: 8 committers face 78 submitters, giving a committer-to-submitter ratio of about 0.1. Six of the eight committers (75%) also appear as submitters. However only three committers are in the list of the top 26 submitters (see Figure 9), the best one at position 13 with 34 patch submissions. Thus, in both projects committers tend to discuss possible changes only seldom on the mailing list.

5. RELATED WORK

While software repositories have been analyzed extensively during the last years, only few researchers have tried to exploit the information provided by email archives. For example, one can construct a social network where nodes represent developers and directed edges with edge weights indicate who sends how many mails to whom [3]. But some emails also contain change information similar to the one stored in software repositories: patches. In his socio-technical study [4], Ducheneaut found out that patches play an important role in the integration of new members in an open-source project. To get access to patches in email archives we basically use the approach first presented by Bird et al. [2], see Sections 2 and 3. Alternatively, one could only look for keywords in the subject of the mails. Asundi and Jayant report a relatively high precision of this simple approach [1], but no information about the presumably low recall is given. In their case study they mainly concentrate on the patch review process itself and address questions such as how many different developers reply to the mail with the patch, or how many replies are sent to a patch mail at all.

6. THREATS TO VALIDITY

To this end we have studied two open-source projects and obtained several interesting results. However, in order to check to what extent these results are valid for other projects, a larger number of software systems should be examined. It should be taken into account, that different projects may have different policies concerning patch submission.

Currently we assume that all detected patches are independent from each other. However, in some cases patches may be discussed, overhauled, and then resubmitted. Thus, patches that are connected in this sense should be marked at least. Furthermore, the patch detection algorithm itself should be evaluated thoroughly concerning its precision and recall.

7. CONCLUSIONS

In this paper we have used a re-implementation of Bird et al.'s techniques [2] to extract patches from emails and find their application in the repository. Furthermore, we have defined when we consider a patch as being accepted.

We applied these techniques to the mailing lists and CVS repositories of two open-source projects. The most important results we obtained for those projects in our case study are the following:

- In both projects about 3–4% of the emails sent to the mailing list contain patches.
- Only about 6% of the files are affected at least once by an accepted patch.
- For both projects the probability that a patch is accepted is about 40%.
- If patches are accepted, they normally are accepted quickly: for FLAC half of the accepted patches have been committed during one week. For OpenAFS even 61% of the accepted patches needed only three days.
- For small patches (at most 4 lines changed) the chances to get accepted are higher than average. For very large patches they are significantly lower than average.
- While the patch size has an effect on the chances to get the patch accepted, it does not influence significantly the duration until the patch is accepted.
- For both projects the number of committers is low compared to the number of submitters (ratio about 0.1–0.2). Although 50% (FLAC) resp. 75% (OpenAFS) of the committers appear at least once as submitters, they tend to send only few patches to the mailing list. Although the number of submitters is large, in both projects a small group of 4–5 core submitters exist who make a specially high number of patch submissions.
- Not all patches sent to the mailing list of a project necessarily are for patches for that project. Instead, patches may address related projects (e.g. to assure or improve compatibility).

As part of our future work, we want to go beyond this and find out if there are specific reasons for some of our findings. Before, we want to address the threats of validity described above, and especially explore more projects.

Additionally, as now e-mail patches are available for us as a new data source, we plan to integrate it into our existing analysis tools [5] to improve the quality of the results.

Acknowledgments

Michael Burch gave helpful comments on earlier drafts of this paper.

8. REFERENCES

[1] J. Asundi and R. Jayant. Patch review processes in open source software development communities: A comparative case study. Los Alamitos, CA, USA, 2007. IEEE Computer Society.

[2] C. Bird, A. Gourley, and P. Devanbu. Detecting patch submission and acceptance in oss projects. In *Proc. 4th Workshop On Mining Software Repositories (MSR07)*, May 2007.

[3] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan. Mining email social networks. In *Proceedings of the 2006 international workshop on Mining software repositories MSR06*, pages 137–143, New York, NY, USA, 2006. ACM.

[4] N. Ducheneaut. Socialization in an open source software community: A socio-technical analysis. *Computer Supported Cooperative Work*, 14(4):323–368, August 2005.

[5] P. Weißgerber, M. Pohl, and M. Burch. Visual data mining in software archives to detect how developers work together. In *Proc. 4th Workshop On Mining Software Repositories (MSR07)*, May 2007.