The Secret Life of Patches: A Firefox Case Study

Olga Baysal, Oleksii Kononenko, Reid Holmes, and Michael W. Godfrey
David R. Cheriton School of Computer Science
University of Waterloo
{*obaysal, okononen, rtholmes, migod*}*@cs.uwaterloo.ca*

***Abstract*—The goal of the code review process is to assess the quality of source code modifications (submitted as patches) before they are committed to a project's version control repository. This process is particularly important in open source projects to ensure the quality of contributions submitted by the community; however, the review process can promote or discourage these contributions. In this paper, we study the patch lifecycle of the Mozilla Firefox project. The model of a patch lifecycle was extracted from both the qualitative evidence of the individual processes (interviews and discussions with developers), and the quantitative assessment of the Mozilla process and practice. We contrast the lifecycle of a patch in pre- and post-rapid release development. A quantitative comparison showed that while the patch lifecycle remains mostly unchanged after switching to rapid release, the patches submitted by casual contributors are disproportionately more likely to be abandoned compared to core contributors. This suggests that patches from casual developers should receive extra care to both ensure quality and encourage future community contributions.**

***Keywords*-Open source software, code review, patch lifecycle.**

I. Introduction

Code review is a key element of any mature software development process. It is particularly important for open source software (OSS) development, since contributions — in the form of bug fixes, new features, documentation, etc. — may come not only from core developers but also from members of the greater user community [1]–[4]. Indeed, community contributions are often the life's blood of a successful open source project; yet, the core developers must also be able to assess the quality of the incoming contributions, lest they negatively impact upon the overall quality of the system.

The code review process evaluates the quality of source code modifications (submitted as patches) before they are committed to a project's version control repository. A strict review process is important to ensure the quality of the system, and some contributions will be championed and succeed while others will not. Consequently, the carefulness, fairness, and transparency of the process will be keenly felt by the contributors. In this work, we wanted to explore whether code review is a democratic process, i.e., contributions from various developers are being reviewed equally regardless of the developers' involvement on a project. Do patches from core developers have a higher chance of being accepted? Do patches from casual contributors take longer to get feedback? OSS projects typically offer a number of repositories or tools to communicate with the community on their feedback. For example, Mozilla provides Bugzilla for filing bug reports or submitting patches for a known defect. Users are also able to leave their feedback (praise, issues, or ideas) for Firefox on a designated website [5].

Improving community contribution and engagement is one of the challenges that many open source projects face. Thus, some projects establish initiatives to improve communication channels with the community surrounding the project. Such initiatives aim at identifying bottlenecks in the organizational process and creating improved eco-systems that actively promote contributions.

For example, the Mozilla Anthropology [6] project was started in late 2011 to examine how various stakeholders make use of Bugzilla in practice and to gain a sense of how Bugzilla could be improved to better support the Mozilla community. During this process, Martin Best — one of the Mozilla's project managers — interviewed 20 community members on their engagement and experience with Bugzilla bug tracking system.

By studying the interview data [7], we noticed that there is a perception among developers that the current Mozilla code review process leaves much room for improvement:

> "The review system doesn't seem to be tightly integrated into Bugzilla." (D10)
>
> "There are issues with reviews being unpredictable." (D11)
>
> "People have different definitions of the review process and disagree a lot, different types of requests". (D16)

Developers also expressed interest in having a version control system on patches, as well as better transparency on code review and statuses of their patches.

This paper provides a case study to evaluate Mozilla's current practice of patch review process and suggests possible improvements to foster community contributions. In particular, we address the following research questions:

Q1: **Do pre- and post-rapid release patch lifecycles for core contributors differ?**
We expect to find differences in how patches are being reviewed before and after Mozilla's switch to rapid release schedule.

Q2: **Is the patch lifecycle different for casual contributors?**
We presumed that code review should not be biased by the contributor's reputation.

Q3: **How long does it take for a patch to progress through each state of its lifecycle?**
We expected to find that patches are being addressed in a more timely fashion in the post-rapid release development model.

The main contributions of the paper are:

1) Development of a model of a patch lifecycle.
2) A case study of a real world code review practice and its assessment using Mozilla Firefox.

The rest of the paper is organized as follows. Section II summarizes prior work. Section III describes Mozilla's code review policy and presents a model of a patch lifecycle. Section IV presents results of the empirical study on evaluation of the patch review process in Mozilla Firefox and Section V discusses our findings on patch review process, and also addresses threats to validity. And finally, Section VI provides final remarks on the main findings.

II. Related Work

Code review processes and contribution management practices have been previously studied by a number of researchers. Mockus et al. [8] were one of the first researchers who studied OSS development. By studying the development process of the Apache project, they identified the key characteristics of the OSS development, including heavy reliance on volunteers, self-assignment of tasks, lack of formal requirement or deliverables, etc.

Rigby and German [2] presented a first study that investigated the code review processes in open source projects, They compared the code review processes of four open source projects: GCC, Linux, Mozilla, and Apache. They discovered a number of review patterns and performed a quantitative analysis of the review process of the Apache project. While they found that 44% of pre-commit patches get accepted, our study shows that on average 78% of patches receive positive reviews. Rigby and German looked at the frequency of reviews and how long reviews take to perform. In our study, we differentiate negative and positive reviews and investigate how long it takes for a patch to be accepted or rejected depending on the contributors participation on a project (frequent vs. occasional).

Later, Rigby and Storey [9] studied mechanisms and behaviours that facilitate peer review in a open source setting. They performed an empirical case study on five open source projects by manually coding hundreds of reviews, interviewing core developers and measuring various characteristics of the OSS review processes.

Asundi and Jayant [4] looked at the process and methodologies of free/libre/open source software (FLOSS) development. They examined the process of patch review as a proxy for the extent of code review process in FLOSS projects. They presented a quantitative analysis of the email archives of five FLOSS projects to characterize the patch review process. Their findings suggested that while the patch review process varies from project to project, the core developers play the vital role in ensuring the quality of the patches that get in. They defined core-group members as those who are listed on the project's development pages. We observed that for Mozilla project some official employees submit very few patches, while developers outside of the core development team provide frequent contributions. Therefore, we employed a different definition of core and casual contributors. In spite of the differences in definitions, our study confirms some of the results reported in [4]. In particular, we also found that core contributors account for a greater proportion of submitted patches and that patches from casual contributors are more likely to be left unreviewed.

Sethanandha et al. [10] proposed a conceptual model of OSS contribution process. They presented the key practices for patch creation, publication, discovery, review, and application and also offered some recommendations on managing contribution process. Bettenburg et al. [11] performed an empirical study on the contribution management processes of two open source software ecosystems, Android and Linux. They developed a conceptual model of contribution management based on the analysis of seven OSS systems. They compared the code review processes of Android and Linux and offered some recommendations for practitioners in establishing effective contribution management practices. While Sethanandha et al. [10] and Bettenburg et al. [11] aimed at developing models of contribution management process, we focus on formalizing the lifecycle of a patch.

Nurolahzade et al. [1] examined the patch evolution process of the Mozilla development community. They quantitatively measure parameters related to the process, explained inner-workings of the process, and identified a few patterns pertaining to developer and reviewer behaviour including "patchy-patcher" and "merciful reviewer". While also studying Mozilla Firefox and exploring patch review process, we modelled the lifecycle of patches rather than the patch evolution process. We also investigated whether the code review is affected by the contributor's participation on a project.

Weissgerber et al. [12] performed data mining on email archives of two open source projects to study patch contributions. They found that the probability of a patch being accepted is about 40% and that smaller patches have higher chance of being accepted than larger ones. They also reported that if patches are accepted, they are normally accepted quickly (61% of patches are accepted within three days). Our findings show that 63% of pre- and 67% of post-rapid release patches are accepted within 24 hours.

III. Patch Review

This section describes the Mozilla's code review policy and presents a model of a patch review process.

A. Mozilla's Code Review Process

Mozilla employs a two-tier code review process for validating submitted patches — *review* and *super review* [13]. The first type of a review is performed by a module owner or peers of the module; a reviewer is someone who has domain expertise in a problem area. The second type of review is called a super review; these reviews are required if the patch involves integration or modifies core Mozilla infrastructure. Currently, there are 29 super-reviewers [14] for all Mozilla modules and 18 reviewers (peers) on Firefox module [15]. However, any person with level 3 commit access — core product access to the Mercurial version control system — can become a reviewer.

Bugzilla users flag patches with metadata to capture code review requests and evaluations. A typical patch review process consists of the following steps:

1) Once the patch is ready and needs to be reviewed, the owner of the patch requests a review from a module owner or a peer. The review flag is set to "*r?*". If the owner of the patch decides to request a super review, he may also do so and the flag is set to "*sr?*".
2) When the patch passes a review or a super review, the flag is set to "*r+*" or "*sr+*" respectively. If it fails review, the reviewer sets the flag to "*r-*" or "*sr-*" and provides explanation on a review by adding comments to a bug in Bugzilla.
3) If the patch is rejected, the patch owner may resubmit a new version of the patch that will undergo a review process from the beginning. If the patch is approved, it will be checked into the project's official codebase.

B. A Model of the Patch Lifecycle

We modelled the patch lifecycle by examining Mozilla's code review policy and processes and compared them to how developers worked with patches in practice. We extracted the states a patch can go through and defined the final states it can be assigned to. Figure 1 presents a model of the lifecycle of a patch. The diagram shows the various transitions a patch can go through during its review process. A transition represents an event which is labelled as a flag and its status reported during the review process.

We considered only the key code review patches having "review" (r) and "supperreview" (sr) flags on them. Other flags on a patch such as "feedback", "ui-review", "checkin", "approval_aurora", or "approval_beta", as well as patches with no flags were excluded from the analysis.

The code review process begins when a patch is submitted and a review is requested; the initial transition is labelled as "r? OR sr?", i.e., a review or super review is requested respectively. There are three states a patch can be assigned

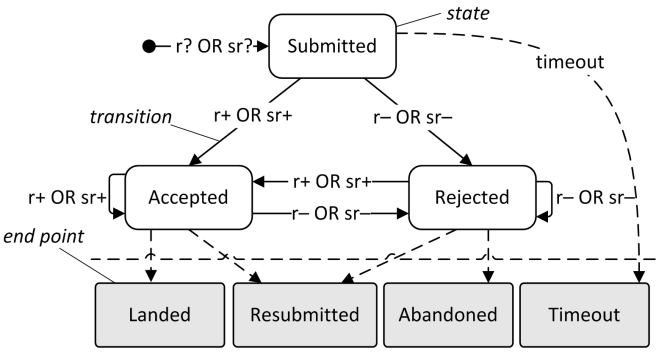

Figure 1. The lifecycle of a patch.

to: Submitted, Accepted, and Rejected. Once the review is requested (a flag contains a question mark "?" at the end), the patch enters the **Submitted** state. If a reviewer assigns "+" to a flag (e.g., "r+" or "sr+"), the patch goes to the **Accepted** state; if a flag is reported with a status "–" (e.g., "r–" or "sr–"), the patch is **Rejected**.

Both the Accepted and the Rejected states might have self-transitions. These self-transitions, as well as the transitions between the Accepted and the Rejected states illustrate the double review process. The double review process takes place in the situations when a reviewer thinks that the patch can benefit from additional reviews or when code modifications affect several modules and thus need to be reviewed by a reviewer from each affected module.

We define and call end points as **Landed**, **Resubmitted**, **Abandoned**, and **Timeout**. These end points represent four final outcomes for any given patch. During the review process each patch is assigned to only one of these four groups:

- Landed — patches that meet the code review criteria and are incorporated into the codebase.
- Resubmitted — patches that were superseded by additional refinements after being accepted or rejected.
- Abandoned — patches that are not improved after being rejected.
- Timeout — patches with review requests that are never answered.

The cumulative number of the patches in Landed, Resubmitted, Abandoned, and Timeout is equal to the number of the Submitted patches.

IV. Quantitative Assessment of the Patch Review Process

In this section, we present our empirical study on the patch contributions within the Mozilla Firefox project. We follow our research questions to evaluate current patch submission and review practice of Mozilla Firefox.

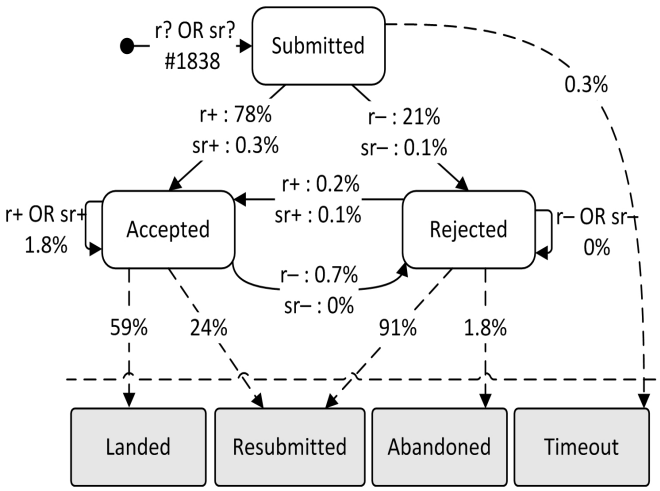

Figure 2. Patch lifecycle for core contributors for pre-rapid release time.

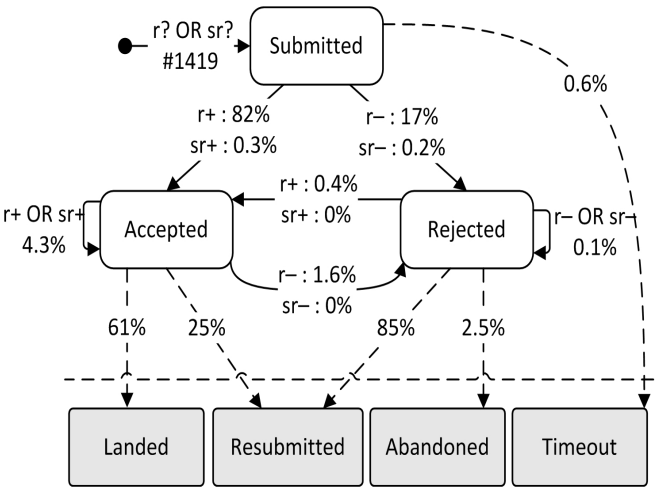

Figure 3. Patch lifecycle for core contributors for post-rapid release time.

Q1: Do pre- and post-rapid release patch lifecycles for core contributors differ?

Approach

On April 12, 2011 Mozilla migrated to a rapid release model (with releases every 6 weeks) from a more traditional release model (with releases averaging every 10 months). We were interested to investigate whether the patch lifecycle of the pre-rapid release differs from the post-rapid release process. Therefore, we compared patch contributions between the following time periods:

- pre-rapid release span: 2010-04-12 to 2011-04-12
- post-rapid release span: 2011-04-12 to 2012-04-12.

One of the key properties of open source projects is that community members are voluntarily engaged in a project. Thus, the amount of the patches received from a particular member varies depending on his interest (and ability) in contributing to the project. The total number of patches submitted in pre- and post-rapid release periods are 6,491 and 4,897 respectively.

We looked at the amount of patches each developer submitted during a two-year timespan and compared various groups of the received contributions. We empirically determined the right amount of contributions to define two different groups of contributors: **casual** and **core**. We compared patch distributions within the lifecycle model for different sets: 10, 20, or 50 patches or fewer for casual contributors; and more than 10, 20, 50, or 100 patches for core contributors. We did not find a significant difference in the distribution of patches for casual developers among 10/20/50 patch sets or for core developers among 10/20/50/100 patch sets. However, to avoid being biased toward the core group when choosing the right threshold for the core group (due to large amount of patches), we decided that 100 patches or more is reasonable to define core contributors. Therefore, we chose to set the casual group at 20 patches or fewer and core group at 100 patches or more. Thus, we considered both the distribution of patches in a patch lifecycle within each group of contributors and the distribution of patches between casual and core groups. The contributors who submitted more than 20 and fewer than 100 patches (12% of all contributors) were out of the scope of the analysis.

Comparison of the Patch Lifecycles

We first compared patch lifecycles for pre- and post-rapid release periods; we considered only patches submitted by core contributors.

Figure 2 and Figure 3 illustrate the lifecycles a patch goes through in pre- and post-rapid release time periods. Edges represent the percentage of the patches exiting state X and entering state Y.

Comparing Figure 2 and Figure 3, we observed the following. Post-rapid release is characterized by the 4% increase in the proportion of patches that get in and the 4% decrease in the percentage of the patches that get rejected. Although the post-rapid release world has a slightly higher percentage of the patches that pass the review and land in the codebase (61% vs. 59% submitted before April 2011), there are more patches that are abandoned (2.5% vs. 1.8%) and twice as many patches that receive no response. After switching to rapid release, developers are less likely to resubmit a patch after it fails to pass a review – a 6% decrease in the number of such patches.

Super reviews are, in general, sparse – only 0.4% (before April 2011) and 0.5% (after April 2011) of the patches are being super reviewed. The proportion of patches that pass super reviews remains unchanged, at 0.3% for both periods; while post-rapid release super-reviewers reject twice as many patches than before.

For post-rapid world, we found that there are two times

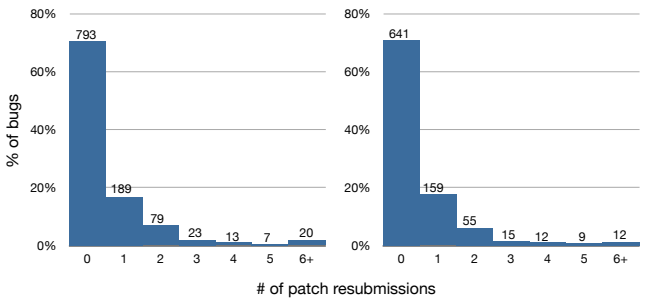

Figure 4. Patch resubmissions per bug for pre- (left) and post-rapid (right) release periods.

more patches that are first rejected but are later accepted and that the proportion of the initially accepted patches that are rejected during the second review phase is increased by a factor of 2.3x, from 0.7% to 1.6%. The situation when a patch goes from r+ to r– occurs quite often in practice. In such a situation, a patch is approved and is checked into the *try* or main tree, but subsequently fails to pass tests on the try server or during the nightly builds. As soon as it crashes the tree trunk it gets flagged "r–".

Patches submitted and accepted after April 2011 are more likely to go through the second round of reviews (4.3% vs. 1.8%). This happens in two cases: 1) when someone else comes up with a better version of a patch or an alternative approach of performing a change; 2) when a reviewer flags the patch as "r+ w/ nits". Such "reviews with nits" are quite common. Reviewers approve a patch even if they are not completely satisfied with the patch. While they comment on what changes (nits), typically small, need to be fixed, they leave it to the contributor to fix them. Such flag assignment requires a certain level of trust between reviewer and developer. "r+ w/ nits" can also happen when a patch requiring small modifications is approved and fixed by the reviewer himself. It is likely that these patches come from casual developers who do not have commit access to checkin the patch themselves.

While the results report some differences in the proportion of patches on each transition of the model in pre- and post-rapid release, the patch lifecycle appears not to have changed much.

Patch Resubmissions

We noticed that 39% (pre-rapid) and 36% (post-rapid) of all patches are being resubmitted at least once. We checked all the bugs including ones with no patch resubmissions. Figure 4 presents two histograms showing the proportion of bugs and the number of patch resubmissions for pre- (left) and post-rapid (right) release periods. There were 1,124 and 903 bugs filed by core contributors during pre- and post-rapid release periods respectively.

The results show that pre and post-rapid release bugs

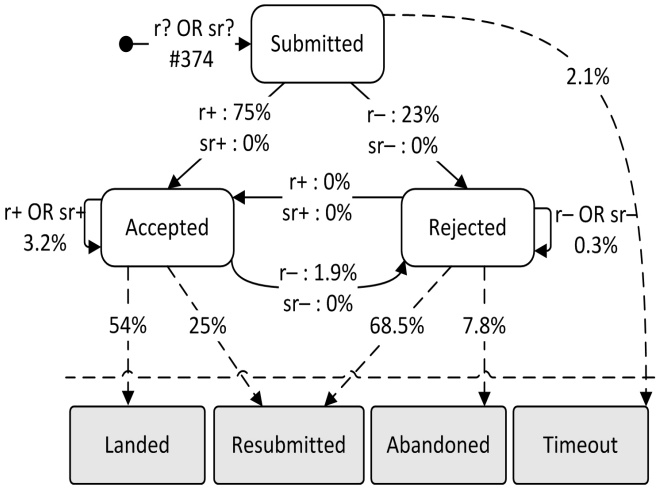

Figure 5. Patch lifecycle for casual contributors.

have a similar distribution of the resubmitted patches. 71% of bugs had no patches that were resubmitted. While 1.8% of pre-rapid release bugs had patches that were resubmitted over 6 times compared to 1.3% of the post-rapid release ones, the highest number of resubmissions in a bug, i.e., 25 patches, is found in the post-rapid release world.

Q2: Is the patch lifecycle different for casual contributors?

Since the lifecycle of a patch for the post-rapid release phase remains almost unchanged, we decided to explore whether there is a difference between patch lifecycles for core (>100 patches) vs. casual (<20 patches) contributors (for a post-rapid release phase only). Since we found no difference in patch lifecycles of casual and core contributors for pre- and post-rapid release periods, we only report the comparison of casual and core contributors for a post-rapid release model to avoid repetition of the results.

Comparing the lifecycles for core (Figure 3) vs. casual contributors (Figure 5), we noticed that, in general, casual contributors have 7% fewer patches that get accepted or checked into the codebase and have 6% more patches that get rejected. The amount of the patches from casual contributors that received no response or are being abandoned is increased by the factor of 3.5x and 3.12x respectively. Review requests with timeouts are likely those that are directed to wrong reviewers or landed to the "General" component that does not have an explicit owner. If a review was asked from a default reviewer, a component owner, the patch is likely to get no response due to heavy loads and long review queues the default reviewer has. Since contributors decide what reviewer to request an approval from, they might send their patch to the "graveyard" by asking the wrong person to review their patch. The process, by design, lacks

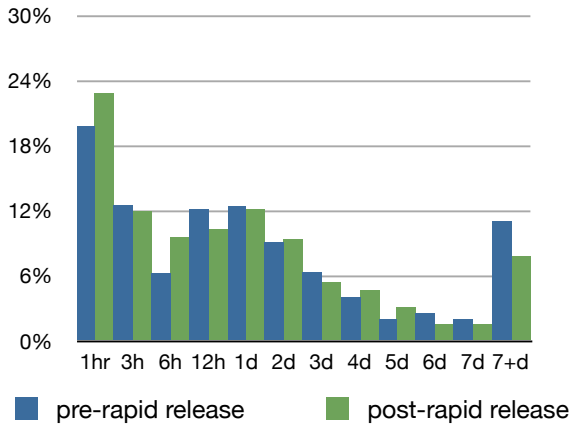

Figure 6. How long does it take for a patch of a core contributor to be accepted?

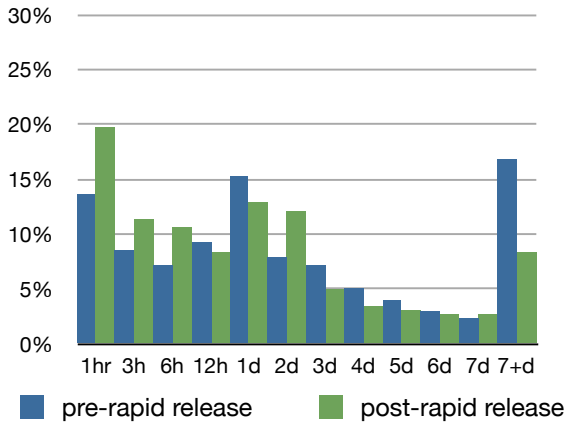

Figure 7. How long does it take for a patch of a core contributor to be rejected?

transparency on the review queues of the reviewers.

Moreover, casual contributors are more likely to give up on a patch that fails a review process – 16% fewer patches are resubmitted after rejection. Unlike patches from core developers, once rejected patches from "casual" group do not get a chance to get in (0% on the "r– →+" transition) and are three times more likely to receive a second negative response.

The results show that patches submitted by casual developers do not require super reviews, as we found no super review requests on these patches. We found this unsurprising, since community members who participate occasionally in a project often submit small and trivial patches [12].

Our findings suggest that *patches from casual contributors are more likely to be abandoned by both reviewers and contributors themselves*. Thus, it is likely that these patches should receive extra care to both ensure quality and encourage future contributions from the community members who prefer to participate in the collaborative development on a less regular basis.

Q3: How long does it take for a patch to progress through each state of its lifecycle?

While previous questions showed the difference in the patch distribution on each transition of the model for both core and casual developers in pre- and post rapid release spans, the model lacks the temporal data on how long these transitions take.

A healthy code review process needs to handle patch submissions in a timely manner to avoid potential problems in the development process [11]. Since a patch can provide a fix to an existing software bug, patches should ideally be reviewed as soon as they arrive.

Since the lifecycle of a patch remains almost unchanged after Mozilla has switched to rapid release trains, we wonder whether patches are reviewed in a more timely fashion.

To answer this question, we analyzed the timestamps of the review flags reported during the review process. We considered only the review flags that define the transitions of the patch lifecycle model. We looked at the time a review flag is added at and computed deltas (in minutes) between two sequential flags in a patch. These two consecutive flags form a flag pair that corresponds to a transition in the patch lifecycle model.

We were interested to determine the time it takes for a patch to be accepted and rejected for core developers in pre- and post-rapid release phases. Figure 6 and Figure 7 report the results.

About 20% and 23% of pre- and post-release patches are accepted within the first hour of being submitted; and 63% and 67% of the overall patches are accepted within 24 hours. Thus, since April 2011 Mozilla has increased the amount of patches that get in within 24 hours by 3-4%, while decreasing the number of patches that require longer attention (over a week review cycle) by 3% (compared to previous 11%).

Mozilla has also increased the amount of patches that receive a negative decision within first hour after switching to rapid release (20% vs. 14%). The number of patches that stay in a review queue for longer than a week is reduced by more than half (from 17% to 8%). One possible explanation is that Mozilla tries to manage the overwhelming number of the received contributions by enforcing reviewers to address patches as soon as they come in.

We then compared the time it takes for patches to be accepted (Figure 8) and rejected (Figure 9) for core and casual contributors.

The main difference is that patches from casual developers are accepted faster, they are more likely to be approved within first hour (26% vs. 23%). Since contributions from casual developers tend to be smaller, this observation is not

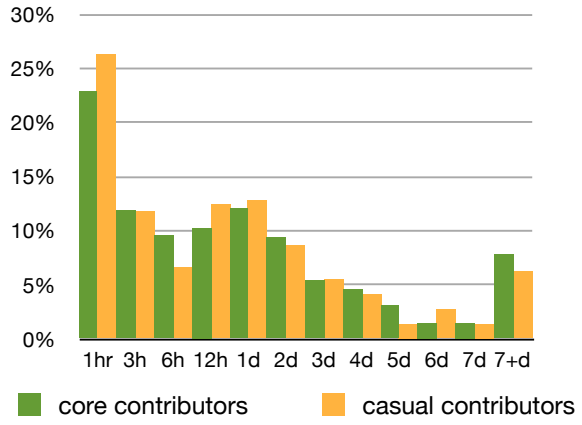

Figure 8. Time until patches from core and casual contributors get accepted in post-rapid release period.

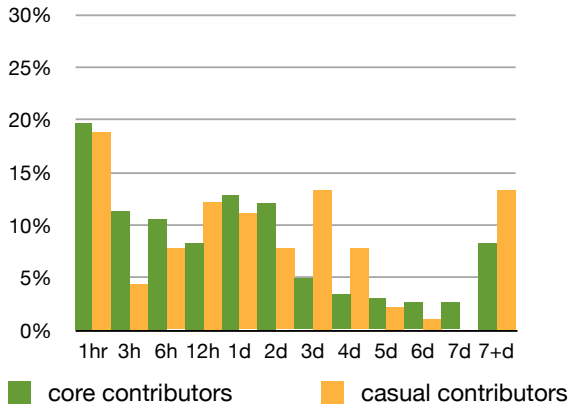

Figure 9. Time until patches from core and casual contributors get rejected in post-rapid release period.

surprising. This finding conforms to the previous research that patches from casual contributors are accepted faster than those from core developers, since reviewers tend to favour smaller code modifications as they are easier to review [3]. In general, the acceptance rate and the proportion of patches for each time interval is consistent among two groups of the contributors.

We observed that *patches from core contributors are rejected faster, around 20% of these patches are rejected within first hour of their submission*. While we did not account for the size of patches, contributions from core developers are generally larger and more complex [2], [4]. By rejecting larger patches early, core developers are notified quickly that their code needs further work to comply with quality standards, thus letting core developers make necessary changes quickly without wasting much time [11]. In contrast to core contributors, casual developers do not receive negative feedback on their patches until a later time, with 13% of all the patches being rejected over a week later. In general, core developers considered as the elite within the community [10] as their contributions are more likely to affect the quality of the codebase and the development of the OSS project. Therefore, it is unsurprising that their patches receive faster negative response. On the other hand, if a patch fails to attract reviewer's attention, the review is likely to be postponed until the patch generates some interest among reviewers or project team members [9].

Lifespan of Transitions

We measured the time it takes for a patch to go from one state to another. Table I reports the median time (in minutes) each transition of the model takes.

Statistically significant results are achieved for transitions "r? → r+" and "r? → r–" when comparing pre- and post-rapid release populations of patches (p-value<0.05). This shows that *after the switch to a rapid release model, decisions on whether to accept or reject a contribution are made faster*.

Table I
THE MEDIAN TIME OF A TRANSITION (IN MINUTES); * INDICATES STATISTICAL SIGNIFICANCE.

Transition	Pre	Post	
		Core	Casual
r? → r+	693*	534*	494
r? → r–	1206*	710*	1024
r+ → r–	1181	390	402
r– → r+	1350	1218	15
sr? → sr+	525	617	n/a
sr? → sr–	6725	9148	n/a

The transition "r? → r+" is a lot faster than "r? → r–" for all three groups such as pre core, post core and casual. This means that reviewers provide faster responses if a patch is of good quality.

To our surprise, the fastest "r? → r+" is detected for casual developers. Our findings show that contributions from casual developers are less likely to get a positive review; yet if they do, the median response rate is about 8 hours (in contrast to 9 hours for core developers).

Super reviews, in general, are approved very quickly, within 8-10 hours. This finding conforms to the Mozilla's code review policy – super-reviewers do provide response within 24 hours of a super review being requested. However, it takes much longer for a super-reviewer to reject a patch requiring an integration. It takes over 4 to 6 days for a super-reviewer to make such a decision, often through an extensive discussion with others.

"r+ → r–" is a lot faster for the rapid release world (6.5 hours compared to 20 hours), meaning previously accepted patches are more likely to be reviewed first during the second round of a review process. A possible explanation for this is that a reviewer might expect that such patches are of better quality and thus they appeal more to him.

In a post-rapid release phase "r– → r+" is a lot slower for core developers, mainly because there is only one occurrence of this transition for the "casual" group.

Lifespan of Patches

In our study, we define the lifespan of a patch to be the period during which a patch exists, i.e., from the time the patch is submitted until the time it is in one of the three final outcomes (Landed, Abandoned, or Resubmitted). Patches with no review responses (in a Timeout state) are excluded from the analysis since they are "undead" until they attract interest among developers.

Table II reports the mean lifespan of patches in each final outcome.

Table II
AVERAGE (MEAN) LIFESPAN OF A PATCH (IN DAYS) FOR FINAL OUTCOMES; * INDICATES STATISTICAL SIGNIFICANCE.

Final Outcome	Pre	Post	
		Core	Casual
Landed	4.5	2.7	2.1
Abandoned	31.2	11.1	7.1
Resubmitted	3.8*	2.5*	4.3

Landed patches submitted by casual developers have the shortest average "life expectancy" at 2.1 days; while abandoned patches submitted by core developers prior April 2011 have the longest average "life" at 31.2 days.

The longest average lifespan in Landed group is for pre-rapid release patches at 4.5 days (compared to 2.7 and 2.1 days for post-rapid release patches). The difference between the average lifespans of the patches submitted by core contributors before and after April 2011 was statistically significant ($p<0.005$). As expected, after switching to a rapid release model, the reviews of patches from core developers are done faster (2.5 days vs. 3.8 days).

Decisions to land patches from casual contributors are made very fast ($\sim$2 days) with 26% of receiving an "accept" within the first hour. Since code modifications from the casual group members are of smaller size and less critical [9], if they are found to be enough interesting or important, they are quickly reviewed. In contrast, if these patches fail to generate interest or have resubmissions, the review decisions take longer ($\sim$4 days).

On average, the lifespan of a pre-rapid release patch is 4.6 days, while the lifespans of post-rapid release patches are 2.8 days and 3.4 days for core and casual developers respectively. Therefore, the shortest-lived Firefox patch is a zero-minute patch submitted before April 2011. The review request on the patch was made and self-approved and the actual time delta was 14 seconds. We also calculated the average lifespan of all the patches in our dataset. The average patch "lives" for 3.7 days, which is about the same as the common mosquito.

V. DISCUSSION

Software projects put considerable effort into defining and documenting organizational rules and processes. However, the prescribed processes are not always followed in practice. Therefore, we tried to detect any disconnects between Mozilla's code review policy and actual practice.

We noticed that two reviewer policies are not being strictly enforced. Only 6.4% of patches from core developers and 5.4% of patches for casual developers are being double reviewed. Mozilla's policies are not consistent due to its commitment to the open communication standards.

We noticed that super reviews happen only rarely. As expected, patches from casual developers do not undergo super reviews as these patches are unlikely to require any integration changes (changes in API, significant architectural refactoring, etc.). One of the Mozilla developers explains:

> "You do not want to do large scale changes. If the codebase is moving fast enough, your Bugzilla patch can go stale in that approach" (D13).

We also looked at single patch contributions and found that while 65% of submitted patches successfully landed to the codebase, 20% of the initial review requests were neglected and 15% of patches were abandoned after receiving "r–" reviews. To promote community contributions, open source projects may wish to be more careful in reviewing patches from the community members who are filing their first patch.

Our findings suggest that rapid release review process offers faster response rates. Within the last year, Mozilla has hired more developers, as well as has started a campaign on providing faster response on contributions (48 hours), in particular for those submitted from the external community members (i.e., not developers employed by the Mozilla Corporation). This shows that Mozilla is learning from the process and being more efficient in managing huge number of contributions from the community.

Threats To Validity

While we performed a comprehensive assessment of the patch review process and practice of Mozilla Firefox, our empirical study is a subject to external validity; we can not generalize our findings to the patch resubmission and review processes of other open source projects. However, the described model of the patch lifecycle is general enough to be easily adjusted by other practitioners should they decide to assess their own process.

Currently we assume that patches (code changes only) within a bug are not independent. We treat consequent patches as resubmissions of the initial patch. In most cases, this assumption holds, an older patch becomes obsolete and a new patch is added to the bug report. We are currently working on the extraction of a richer dataset using the Elastic Search database, a JSON based search query engine that is

mapped to Bugzilla but stores only the meta-data of a bug. Mozilla's intention is to make the Elastic Search database publicly available; however, it is currently in a testing stage.

Our study considers only the review flags such as "review" and "super review". A possible extension of the work would be to include other flags into the analysis such as "feedback", "checkin", and "approval" flags. For example, by mapping "checkin" and "approval" flags to the projects' version control system, we could evaluate project's release management process and practice. Unfortunately, these flags are often not accurately set in practice or are not reported by the project's release drivers.

VI. Conclusions

The code review process is a key part of software development. Like many open source projects, the codebase of Mozilla Firefox evolves through contributions from both core developers as well as the greater user community. Our findings show that Mozilla's effort to reduce the time patches spend waiting for reviews after switching to the rapid-release model was successful. We observed that review of patches from core contributors follows "reject first, accept later" approach, while review of patches from casual developers follows "accept first, reject later" approach. While the size of contributions might account for this difference, further research is needed. While large portion of submitted patches is accepted by reviewers, there is the risk of alienating valuable contributors when reviewers do not respond to the submitters of received contributions. This is particularly important for the patches submitted by casual developers. Creating a first positive experience with the review process for community members is important to encourage possible future contributions from them.

This paper proposes a model of a patch lifecycle that can be used to assess the code review process in practice.

The main findings of our empirical study are as follows:

- Since switching to rapid release, patches are reviewed more quickly.
- Contributions from core developers are rejected faster.
- Contributions from casual developers are accepted faster.
- Patches submitted by casual contributors are disproportionately more likely to be abandoned compared to core contributors.
- Patches "live" for 3.7 days on average.

We hope that our research findings can help inform the decisions made by various practitioners, including Mozilla Corporation, in improving their patch management practices.

Acknowledgement

We wish to thank Martin Best and David Mandelin from Mozilla for their feedback on the paper. We are most grateful to Martin Best for fruitful conversations on Mozilla's code review and project management practices.

References

[1] 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," in *Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops*, 2009, pp. 9–18.

[2] P. Rigby and D. German, "A preliminary examination of code review processes in open source projects," University of Victoria, Canada, Tech. Rep. DCS-305-IR, January 2006.

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

[4] J. Asundi and R. Jayant, "Patch review processes in open source software development communities: A comparative case study," in *Proceedings of the 40th Annual Hawaii International Conference on System Sciences*, ser. HICSS '07, 2007, pp. 166c–.

[5] Mozilla, "Firefox Input," http://input.mozilla.org/, 2012.

[6] M. Best, "The Bugzilla Anthropology." [Online]. Available: https://wiki.mozilla.org/Bugzilla_Anthropology

[7] O. Baysal and R. Holmes, "A Qualitative Study of Mozillas Process Management Practices," David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada, Tech. Rep. CS-2012-10, June 2012. [Online]. Available: http://www.cs.uwaterloo.ca/research/tr/2012/CS-2012-10.pdf

[8] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of open source software development: Apache and mozilla," *ACM Trans. Softw. Eng. Methodol.*, vol. 11, no. 3, pp. 309–346, Jul. 2002.

[9] P. C. Rigby and M.-A. Storey, "Understanding broadcast based peer review on open source software projects," in *Proceedings of the 33rd International Conference on Software Engineering*, 2011, pp. 541–550.

[10] B. D. Sethanandha, B. Massey, and W. Jones, "Managing open source contributions for software project sustainability," in *Proceedings of the 2010 Portland International Conference on Management of Engineering & Technology*, July 2010.

[11] N. Bettenburg, B. Adams, A. E. Hassan, and D. M. German, "Management of community contributions: A case study on the android and linux software ecosystems," 2010. [Online]. Available: http://nicolas-bettenburg.com/?p=399

[12] P. Weissgerber, D. Neu, and S. Diehl, "Small patches get in!" in *Proceedings of the 2008 international working conference on Mining software repositories*, 2008, pp. 67–76.

[13] Mozilla, "Code Review FAQ," https://developer.mozilla.org/en/Code_Review_FAQ, 2012.

[14] ——, "Code-Review Policy," http://www.mozilla.org/hacking/reviewers.html#the-super-reviewers, June 2012.

[15] MozillaWiki, "Modules Firefox," https://wiki.mozilla.org/Modules/Firefox, June 2012.