Broadcast vs. Unicast Review Technology: Does it Matter?

Armstrong Foundjem, Foutse Khomh, Bram Adams
SWAT–MCIS, École Polytechnique de Montréal, Québec, Canada
{a.foundjem, foutse.khomh, bram.adams}@polymtl.ca

***Abstract*—Code review is the process of having other team members examine changes to a software system in order to evaluate their technical content and quality. Over the years, multiple tools have been proposed to help software developers conduct and manage code reviews. Some software organizations have been migrating from broadcast review technology to a more advanced unicast review approach such as Jira, but it is unclear if these unicast review technology leads to better code reviews.**

This paper empirically studies review data of five Apache projects that switched from broadcast based code review to unicast based, to understand the impact of review technology on review effectiveness and quality.

Results suggest that broadcast based review is twice faster than review done with unicast based review technology. However, unicast's review quality seems to be better than that of the broadcast based. Our findings suggest that the medium (i.e., broadcast or unicast) technology used for code reviews can relate to the effectiveness and quality of reviews activities.

***Index Terms*—Code Review, Patches, Apache, Empirical Study, Medium, Bug-inducing Changes , Quality Assurance**

I. Introduction

Peer Code Review (PCR) also known as Code Review (Review) is an integral part of the quality assurance process of software projects [1] [2] [3]. The main objective of Review is to systematically examine source code changes (SCC) written by other programmers in order to verify their correctness and quality before integration into the Version Control System (VCS) (e.g., Git).

While the formal code inspection activities proposed by Fagan [4] in 1976 were heavyweight, nowadays the software industry has adopted lightweight variants of Review, supported by modern tools and technologies [5] [6] [7]. In order to improve quality assurance [2], recent research has even proposed automatic Review tools [8] to reduce human efforts [9].

Currently, there are two main categories of technologies for Reviews. The first category uses a broadcasting technology approach of communication whereas the second category uses a unicast technology approach of communication.

- In the broadcast technology approach, when a patch is submitted for review, all those who are subscribed to the medium where the patch is submitted can see the patch see Figure 4, and anyone can review the patch. Furthermore, any subsequent discussion concerning the patch is visible to all the members who are subscribed to the medium; hence broadcasting. An example of a broadcast technology is the developer's mailing-list [10] [11].
- In the unicast technology approach, when a patch is submitted for review, it is visible only to a targeted (a set of dedicated) group of individuals who are subscribed to the communication medium and who have been assigned to the patch for review see Figure 5. Only those targeted individuals can review the patch and followup discussions concerning the patch. An example of the unicast approach is an issues tracker [12], such as Gerrit, or JIRA etc. In this paper we focus on JIRA.

Many software development organizations such as the Apache Software Foundation (ASF) are migrating from the broadcast technology approach review environments [13] [14] [15] to the unicast technology approach [5] [6] [16] [17] issues tracker environments such as Gerrit, Jira and the like. However, despite this keen interest in the unicast technology Review platforms, there is no evidence that they make code review activities better. Although there have been many studies on code review [5] [18] this past decade, to the best of our knowledge, none of these studies have performed a head-to-head comparison based on the technology used for reviewing code, in terms of effectiveness, effort, and efficiency of Review activities.

This paper aims to fill this gap. More specifically, we aim to examine if the technology (i.e., broadcast vs. unicast) through which patch reviews are performed relates to effort, effectiveness and efficiency of review activities. In order to achieve this goal, we use data from five open source projects from the Apache software foundation (i.e., Flex, HBase, Hive, PIG, and SVN), that have switched from broadcast technology approach to unicast technology approach (issues tracker), which in this case is JIRA. We address the following research questions (**RQs**):

RQ1 - Is review effort related to the review medium used?
Result: We measure the review effort for a patch using information about the number of developers involved in the review, the number of rounds necessary to review the patch, and the number of review requests for the patch. Our results show that in general, patches reviewed on unicast technology undergo more revisions than patches reviewed on broadcasts.

RQ2 - Is the effectiveness of a patch reviewing process related to the medium used?
Result: To capture the effectiveness of a review process, we compute the proportion of reviewed patches that experienced a post release bug, using the SZZ algorithm [19]. Results show

that patches reviewed through broadcast technology are more likely to experience a post release bug than those reviewed on unicast technology. We examined the effect of size on this result and found no significant difference between the sizes of patches reviewed respectively through broadcast and unicast technology.

RQ3 - Is the efficiency of a patch review process related to the medium used?

Result: We measure the efficiency of a patch review process using two metrics: the response delay in days after a patch review request is submitted and the period of time between the submission of a patch review request and the integration of the patch into the VCS. Results show statistically significant differences between the response delay (RD) and the review length (RL) respectively, of patches reviewed on broadcast technology and those reviewed on unicast technology. However, we found no correlation between the size of patches against the response delay and the size of patches against the length of the review period (RL).

The rest of this paper is organized as follows: Section V reviews the related literature, Section II describes the approach used to collect, filter, compute and analyze data. In section III, we present and discuss the results of our research questions. We discuss the threats to validity of our study in section VI. Finally, we conclude the paper in section VII.

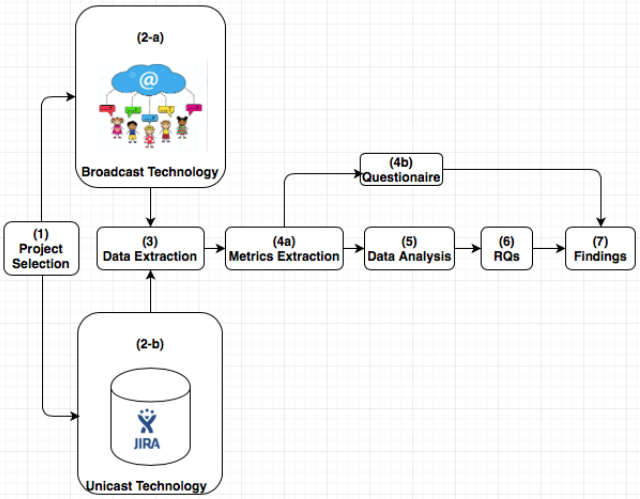


Figure 1: Our Approach.

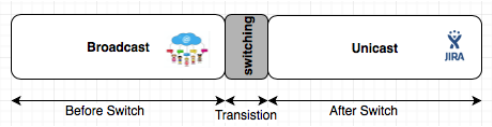


Figure 2: Our Approach: time-line.

II. Approach

In this section, we present the steps of our approach. First, we select stable Apache projects that were using the broadcast technology approach for reviewing patches and at some point in time, switched to a unicast technology approach for reviewing patches. Then, we collect and analyze the data required to answer our research questions. We also interviewed reviewers of the Apache Software Foundation (ASF), using a questionnaire [20] that we designed. Figure 1 shows an overview of our approach and Figure 2 illustrates the time periods that are considered in this study. The middle gray region indicates the window (threshold) period where the switched from broadcast technology approach to unicast technology approach occurred.

A. *Projects Selection*

The Apache ecosystem has projects that were using the broadcast technology approach for reviewing patches and later on switched to unicast technology approach, moreover, Rigby et al. [21] [22] have studied the Apache's broadcast based reviews, and Bettenburg et al. [23] have also studied projects in the Apache Ecosystem. Therefore, the Apache ecosystem appeared to be the ideal choice for our study. To identify the period during which the projects transitioned from broadcast technology approach to unicast technology approach, we examined the volume of patches reviewed on both platforms (broadcast and unicast technologies) over a given period of time and observed that in the gray region shown in Figure 2, there were a drastic decrease in the number of patches that were processed using the broadcast technology approach and a sharp increase of the number of patches processed using the unicast technology approach. The intersection point where broadcasts crosses over unicast forms the transition point for the projects; an example is shown by the gray region in Figure 2 and Figure 3 for the PIG project.

To identify the transition period of the PIG project for example, we computed the proportion of patches reviewed on broadcast platforms and the proportion of patches reviewed on unicast platforms between January 2012 to December 2012. We observed that initially, reviewers reviewed between 33 to 42 patches through the broadcasts each week, in the 25^{th} week we observed a sudden decrease of patches, that is, below 10 patches per week over a period of 6 months (July to Dec), see Figure 3. At the same time unicast's reviewers were experiencing a drastic increased of patches reviewed from about 10 to 35 patches weekly, see Figure 3.

Based on our observation, we defined a window period (the rectangle in Figure 3), which also represents the rejection window, i.e., we reject all the patches within this period for this study. In the case of the PIG project for example, we consider broadcast patches up-to the left boundary of the window (25^{th} week) and unicast patches from the right boundary (30^{th} week) onward to the limiting period and do the same for other projects in this study. Within this window, that is between the (25^{th} and 30^{th}week), since there is an overlap between patches using broadcast and patches using unicast, we rejected all the patches within this window's region so as to have a clean dataset. This window takes into account the period immediately before and immediately after switching from broadcast to unicast.

To determine the window, considering the PIG project for example, we computed the intersection point of the two curves (i.e., the proportion of patches reviewed on broadcasts and the proportion of patches reviewed on unicast), and uses it as a reference point to know exactly when the project switched from broadcast to unicast.

This reference point is based on the volume of previously submitted patches that were reviewed using broadcasts (period before switch) and patches reviewed using unicast (period after the switched). For the PIG project, the reference point is on the 27th and a half weeks, which corresponds to the transition period between July and August. As can be seen from Figure 3. Then, we observe the period when the patches drastically decrease when using broadcast; the left boundary of the window (25th week), and the period when patches drastically increase when using unicast; the right boundary of the window (30th week), this gives us the window size of a project (PIG project), with a cutoff of less than 10 patches per week (the intersection point where broadcast crosses over unicast). Five projects of the Apache ecosystem had reference points where reviewers did less than 10 patches per week within this window that we defined. Therefore, we selected these projects for our study: HBase, Hive, PIG, SVN and Flex.

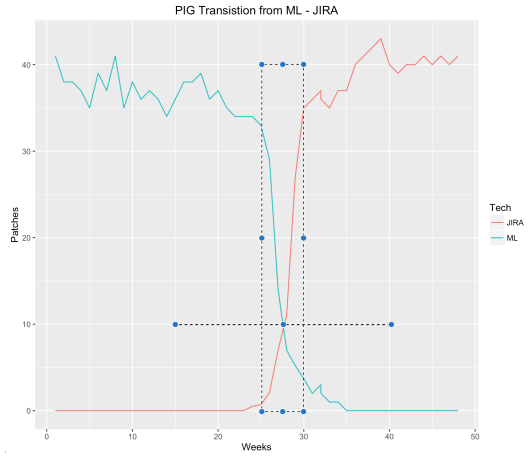


Figure 3: PIG Transition from broadcast to unicast

B. *Broadcast Technology vs. Unicast Technology Approach*

After selecting the projects needed for this study, our next step was to understand the structures of both the broadcast and unicast approaches of reviewing patches.

1) Broadcast Technology Approach: broadcasts archives are arranged as *.mbox* files [24], categorized per year, thread, date and author, with the associated number of threads. This structure is the same for the broadcast archives of all the five projects. Also, the patches are attached to the emails. We use a python script to download the *.mbox* files for all the five projects within the given period of our study as shown in Table III.

2) Unicast Technology Approach: Issue tracker [25] are also areas where source code changes are reviewed before committing to the code-base repository. Moreover, they are enablers for the review process in the sense that, they capture comments regarding the proposed changes of the source code and enable ongoing discussions on the topic. Since all commits have a specific identification number (ID), issue tracker systems use this ID to link revisions to commits, in addition, they keep track of each revision, the author, date, and history of the code changes.

Most issue tracker environments, like unicast, provide web portal to visualize the reviewing process and by default, groups every revision on the same page. Figure 5 (courtesy of Atlassian [1]) shows the work flow and status of the code review process using different colors to indicate the status.

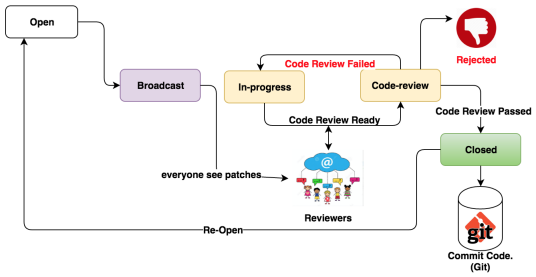


Figure 4: Work-flow and Status in broadcast environment.

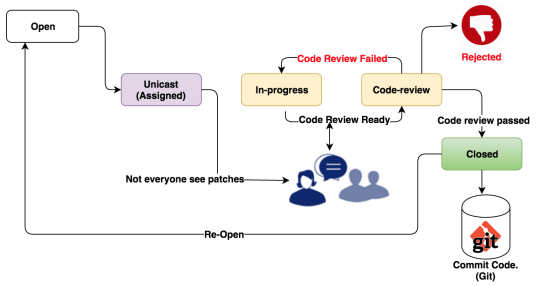


Figure 5: Work-flow and Status in unicast environment.

C. *Data Extraction*

After identifying the data sources for both broadcasts and unicast, we used the following tools to extract the data as needed and created a database to store the data.

Broadcast Technology: We used *MailboxMiner*2 [26] (a Java application) to create a relational postgreSQL database with tables containing the email messages and their metadata. We analyze email attachments, to extract information about the patches.

Emails are grouped per thread, where each email is a reply to an earlier email. Each thread contains one or more emails with different revisions (versions) of a code patch, hence, we need to distinguish the emails with patch revisions from those with review comments (the other emails).

Basically, since we need all revisions of a patch, we used the identification number of patches to group all the thread of

[1]https://www.atlassian.com/software/jira

the same patch. For example, this attachment from an email: 789.*patch* has a key word in the message body HBASE-789, we used this keyword to link patches to Git [27]. In the end, for each patch sent to a broadcast, we can reconstruct all revisions of the patch as well as their general comments. Of course, since emails can contain any kind of text, we need to eliminate noise, such as a reply that does not contribute anything to the Review. However, emails automatically generated by unicast have a consistent pattern attached to them, which allows to identify a particular commit. For example, HBase-960, means a commit from the HBASE project number 960. Same for PIG-XXX, Flex-XXX, and SVN-XXX, and if there is a reply from a reviewer, the message will contain "RE:". With the help of this specific Identification, we were able to link email contributions to the Git repositories [23] with the help of this mirror site [27]. We used git commands such as git diff, git blame, and git log to extract various information about the patches reviewing process.

Unicast Technology: We could extract and export data from unicast in Excel format. We wrote a Python script to process the data.

D. Structure of the Questionnaire

We interviewed reviewers from the Apache Software Foundation using a questionnaire[2], containing both open-ended (aims at asking the reviewers to formulate their own answer) and closed-ended (aims at asking the reviewers to chose an answer from a set of given options) questions. We arranged the questions logically; connecting questions from one to the next, from a general to a more specific perspective. In some of the questions, we wanted to know what motivated the switch from broadcast to unicast review technology, and the difficulties that the reviewer faced in changing their review process. Moreover, we go further to know details about the review process, based on their individual experiences.

After designing the questions, we sent out the questionnaire form [20] to selected Apache reviewers. The procedure to select the Apache reviewers to participate in this survey is as follow. First, for the five projects we used in this study, we selected the top 10 active reviewers per project (participants must have contributed in reviewing patches). These 10 active reviewers per project gave us 50 reviewers in total. Then we filtered 50 reviewers by selecting those who used the broadcast to review patches and then continue reviewing using unicast over the study period. One of the project (SVN) had seven (7) of such reviewers in total, so we decided to normalized all the projects to seven top reviewers per project. That is how we ended up with 35 reviewers (7-reviewers x 5-projects) in total for our questionnaire.

This selection ensures us that the surveyed reviewers can give us meaningful feedback, which would represent the entire period of our study.

[2]https://goo.gl/forms/AgQMasO7mne6G2jg1

Table I: Metrics used to captured review effort, effectiveness, and efficiency.

Type	Metric	Description
Effort	Number of Reviewers (NR)	**N**umber of developers who Participated in the review of a patch
	Number of revisions (NV)	**N**umber of rounds necessary to review a patch [1]
	Review queue (RQu)	**N**umber of pending reviews request per reviewer; reviewer workload [28]
Effectiveness	Post review bugs	**P**roportion of reviewed patch that experienced a post release bug. To identify a bug introduced after a patch review, we use the **SZZ** algorithm [19]
	Median review rate (MRR)	**T**his is the number of lines of codes (KLOC/week) for each review (also known as the Average review rate) [1] [30] [31]. Since we can't claim that the review rates are symmetrically distributed, we will use the median review rate instead. *The median review rate metric aims to examine if reviewers are maintaining a good code-reviewing cadence, for example, not going above 400KLOC/hr or not performing below 200LOC/hr* [30] [32].
Efficiency	Review length (RL)	**T**ime (days) from submission of a review request to the integration of the patch [1]
	Response Delay (RD)	**N**umber of delay in days after a patch review request is submitted [1]
	Size	**S**ize of a patch [28]

E. Metrics Extraction

Based on our RQs and existing work [1] [28] [29], we extracted the metrics presented in Table I. The metrics are grouped according to the RQs.

RQ1 - We capture the effort required to review patches, see Table I.

RQ2 - In this RQ, we defined effectiveness as the ability of reviewers to detect bug-inducing commits during code reviews. We use the SZZ-algorithm [19] [33] [34], that automatically identifies fix-inducing commits by linking bugIDs found in commit messages with bugs repository using keywords such as "FIXED" and using regular expressions. This algorithm back tracks lines changed through the revision history to the point of its most recent change. After locating the keywords, we use the diff tool to know what changed in bug-fix, and outputs a hunk. In addition to tracking post review bugs, we also computed the median review rate for each Review platform (in KLOC/week) [31].

RQ3 - We capture the efficiency of a review process using the review length and the response delay metrics described in Table I. To compute the response delay, we calculate the difference from the reply date of the first revision. To capture

the review length, we compute the time period between the submission of a review request and the integration of the reviewed patch into the version control system.

F. *Data Analysis*

To address our research questions (RQs), we test the following null hypothesis for each metric m:
H_0 *: There is no significant difference between the value of metric m for patches reviewed on broadcasts and those reviewed on unicast.*
We use the Mann-Whitney U test to test H_0. The Mann-Whitney U test is a nonparametric statistical test used for assessing whether two independent distributions have equally large values. Nonparametric statistical methods make no assumptions about the distributions of the assessed variables. We set α to 0.05. Moreover, we consider a Mann-Whitney test result to be statistically significant if and only if the p-value is below α. Additionally, we compute the effect-size of the difference using Cliff's δ (cohen 1998) [35], which is also a non-parametric statistic test, effect size measure, which measures how often values in one distribution are larger than values in another distribution. Cliff's δ d values lies between [-1,1] and is considered negligible (N) for |d| <0.147, small (S) for $0.148 \leq$ |d| < 0.33, medium (M) for $0.33 \leq$ |d| <0.474, and large (L) for |d| $\geq$ 0.474. Our results are reported in Section III below.

Furthermore, we used the Spearman's rank correlation coefficient (Spearman's ρ, $-1 \leqslant \rho \leqslant 1$) to compute the correlation among the metrics (size, RD, and RL) used in this RQ, that is, we computed the ρ, between the size of patches and the responds delay and found it to be 0.063 (negligible positive correlation), between the size of patches and review length and found it to be 0.091 (negligible positive correlation), also between response delay and review length and found it to be 0.168 (negligible positive correlation).

Spearman's rank correlation coefficient is a nonparametric measure of rank correlation that measures statistical dependence between the ranking of two variables. It assesses how well the relationship between two variables can be described using a monotonic function. Mukaka et al. explain how to interpret the values of ρ [36].

III. Results

In this section, we present and discuss the results of our research questions. For all values of the studied metrics that are statistically significant, unicast values on average are higher than those of broadcast.

RQ1 - Is review effort related to the review medium used?
Motivation: Code review is a human activity that requires effort [29][37] from reviewers. The more reviewers contribute to a patch, the less likely it will be buggy. In fact, studies show that less intensively scrutinize files turn out to be defective in the future [1]. In this research question, we want to understand if the effort put on reviewing a patch can be dependent of the reviewing medium.
Approach: To answer this research question, we use the following measurements of reviewers' effort: number of reviewers, number of revisions, and size of the review queue. To compute the number of revision for broadcast, we counted the number of times each distinct email is involved in the revision process of the patch. The Reviewers' activities overtime is shown on the survival curve, see Figures 6.
Findings: *Patches reviewed on unicast undergo more iterations (revisions) than those reviewed on broadcasts, and unicast's reviewers are more active during code review.*

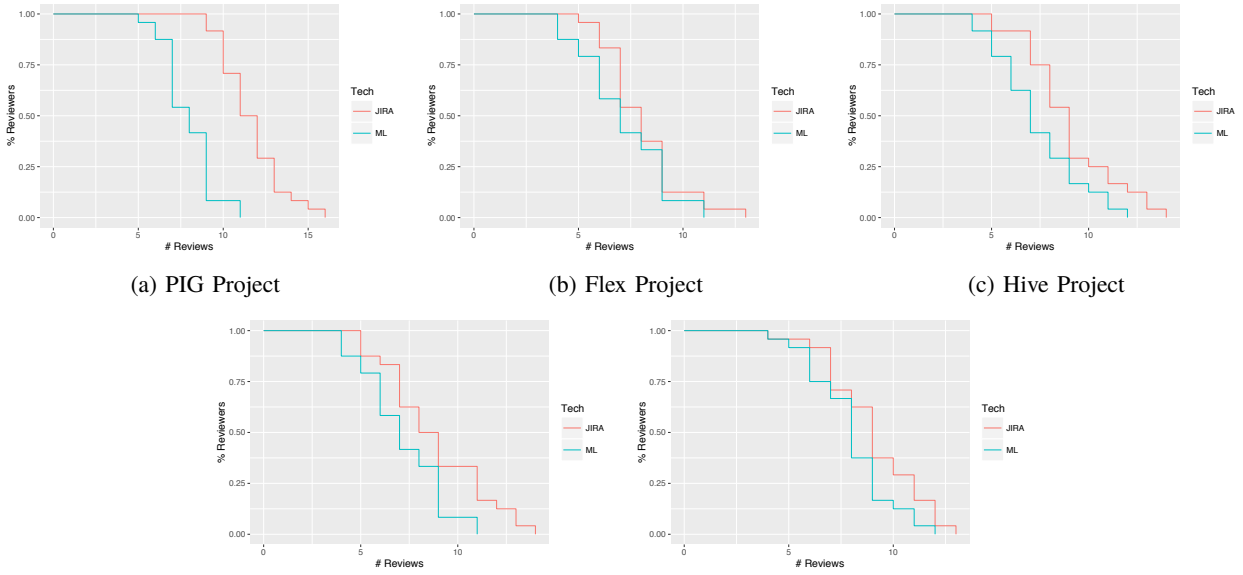
The number of reviewers actively reviewing patches for both system is shown in the survival curve on Figure 6. We considered all the selected projects for both systems. Notice that all the curves starts from 100% (percentage of active reviewers) and decreases as time goes on. On the y-axis, we have the reviewers and on the x-axis, number of reviews that are surviving. Based on the distribution of the survival curve on Figure 6, we considered reviewer's survival rate at 100%, 50% and 20% of active reviewers as shown on Table IV. 100% of reviewers is the initial point, we will like to know how involved the reviewers are at this point and also at half way down 50% and also at 20%. The Table IV below gives a summary of our analysis of reviewers and the number of patches reviewers are reviewing. We observed that 20% of reviewers in unicast can handle a median of 11 patches whereas the same amount of reviewers for broadcast can review a median of 9 patches. That means, reviewers on unicast survive longer than those on broadcast, hence they put in more efforts on patch reviewing process.

> There is a statistically significant difference between the number of patch iterations and the size of the review queue for broadcasts and unicast patches.

RQ2 - Is the effectiveness of a patch reviewing process related to the medium used?
Motivation: The main purpose of code review is to identify errors or mistakes that were made by the developers, therefore, a reviewer should assure that the code is reliable and not error prone, unfortunately, reviewers some times fail to properly examine codes written by developers. In this research question we examine the effectiveness of reviews performed on broadcast based platforms and unicast based platforms, to understand if developers catch more errors on broadcast based platforms or unicast based platforms.
Approach: We apply the SZZ algorithm to map reported bugs to reviewed patches. We ran the SZZ algorithm on the dataset of each project and identify all reviewed patches that induced a future bug. Next, we computed the number of bugs found when using broadcast and when using unicast technology. We also computed the median review rate on broadcasts and unicast respectively. To control for the effect of patch size on our results, we also computed and compared the size of patches (as shown on Figure 8), which are reviewed on broadcasts and unicast.

Table II: p-value and Cliff's-δ results for our studied metrics referred in table I.

Projects	NR		NV		RQu		MRR		RL		RD		Size		SZZ	
	p-value	Cliff's δ	p-value	Cliff's δ	p-value	Cliff's δ	p-value	Cliff's δ	p-value	Cliff's δ	p-value	Cliff's δ	p-value	Cliff'sδ	p-value	Cliff'sδ
Flex	0.843	N	**0.016**	L	**0.012**	L	**0.021**	L	**0.015**	L	**0.019**	L	0.924	N	**0.015**	M
HBase	0.891	N	**0.021**	L	**0.017**	L	**0.017**	L	**0.013**	L	**0.016**	L	0.831	N	**0.039**	M
Hive	0.902	N	**0.011**	L	**0.031**	L	**0.014**	L	**0.021**	L	**0.011**	L	0.913	N	**0.031**	M
PIG	0.931	N	**0.017**	L	**0.021**	L	**0.011**	L	**0.015**	L	**0.012**	L	0.917	N	**0.017**	M
SVN	0.915	N	**0.013**	L	**0.019**	L	**0.031**	L	**0.017**	L	**0.022**	L	0.821	N	**0.008**	M



(a) PIG Project (b) Flex Project (c) Hive Project

(d) Hbase Project (e) SVN Project

Figure 6: Review survival curve for HBase and SVN projects.

Table III: Projects transitional periods and reviews.

Projects	**Study-Period**	**Transition**	**Reviews unicast - broadcasts**
Flex	2011-2015	11-2013	193 - 195
HBase	2011-2015	08-2013	212 - 222
Hive	2011-2015	05-2013	216 - 177
PIG	2011-2015	07-2013	280 - 291
SVN	2010-2014	10-2012	183 - 179

Table IV: X percent of reviewers did at least Y reviews.

Medium	pct	**Projects (number of reviews)**				
		Flex	HBase	Hive	PIG	SVN
broadcasts	100%	4	4	4	5	5
	50%	7	8	7	10	8
	20%	9	9	9	12	9
unicast	100%	5	5	5	7	9
	50%	8	9	9	12	12
	20%	9	11	11	20	13

Findings: *The proportion of reviewed patches that experienced a future bug is lower for unicast than broadcast.*

To ensure a fair comparison between patches reviewed on both technologies; broadcasts and unicast, we filtered the data as mentioned in RQ1 to have the same time window before and after, and observe post review bugs for each patch. After that, we varied the size of this time window and computed on each window the accepted patches that experienced a future bug. Unicast technology Reviews are more effective in terms of identifying buggy patches than reviews done on broadcast technology. Figure 7 presents the distributions that we obtained. The proportion of reviewed patches that experienced a future bug is lower for unicast. We also report results of the MRR in Table II, which is higher for broadcast than unicast. Overall, there is a statistically significant difference between the MRR on broadcasts and unicasts, hence, we reject our null hypothesis for this metric indicating that unicast is better than broadcast in terms of effectiveness.

> Reviews performed on unicast technology are more effective in terms of catching bugs than those performed on broadcast technology.

RQ3 - Is the efficiency of a patch review process related to the medium used?

Motivation: Early studies showed that patch review is time consuming and slow [38]. Studies also show that participation in code review influences quality [28]. This research question analyzes whether the medium on which reviews are conducted relates to the efficiency of the patch review process.

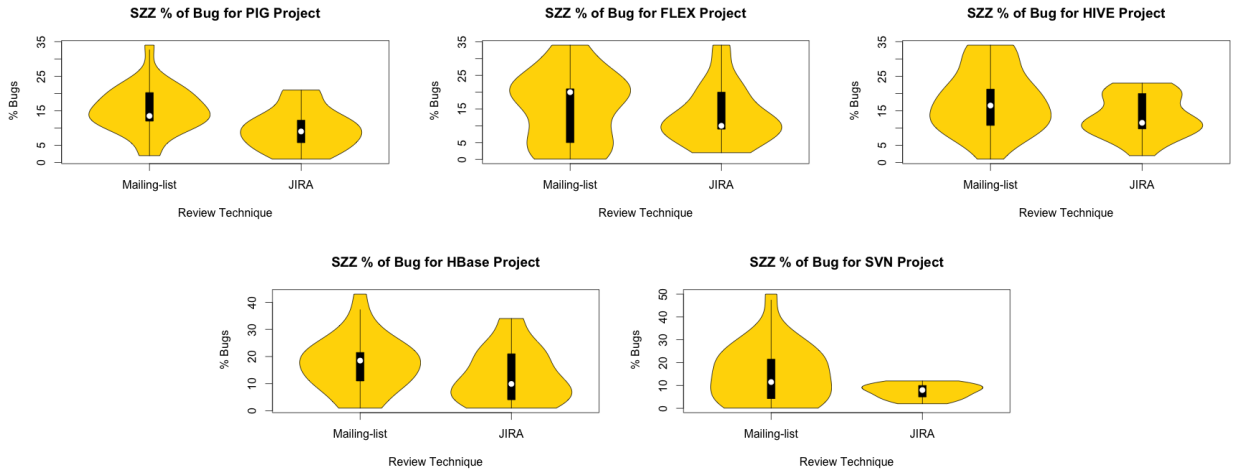


Figure 7: Proportion of reviewed patches that experienced a future bug for different time windows.

Approach: To answer this research question, we compute two metrics: the review length (days) (RL), and the response delay (RD), see Table I. To control for the effect of patch size on our results, we also computed and compared the size of patches reviewed on broadcast and unicast technologies.

Findings: *There are statistically significant differences between the response delay and the review length respectively of patches reviewed on broadcast and those reviewed on unicast. Broadcast has a short response delay and a shorter review length.*

Results presented in Table II show that there is a statistically significant difference between the review length (RL) of patches on broadcast and unicast platforms, and also for the response delay (RD). Moreover, the median value of the review length is lower on broadcasts (i.e., 20 days, which is less than half the value of the median for unicast).

Moreover, there is no statistically significant difference between the size of patches reviewed on both broadcast and unicast technologies, therefore, size is not the factor behind the observed differences in *response delay* and *review length*.

We compared the sizes of patches reviewed on the two platforms (i.e., broadcast and unicast) because the time required to review patches could depend on the size of patches under review.

Our results also show that patches are reviewed on broadcast platforms faster than on unicast platforms (in terms of review length and response delay). This is irrespective of the size of the patches.

We found no difference between the size of patches reviewed on broadcast and unicast platforms. We also found no correlation between the size of patches and the response delay, or the size of patches and the review length. However, we found a statistically significant difference between the response delay (and the review length) of patches reviewed on broadcast platforms in comparison to those reviewed on unicast platforms.

> Overall, we conclude that the medium on which code review is performed can affect the efficiency of the review process.

A. Selected Results from Survey.

To analyze the response from the Apache reviewers who answered our questionnaires, we grouped the response (A) of each question (Q) in the questionnaire and then did a summary per question. Then, we compared them with the results of our analysis.

Among the 35 copies of the questionnaires we sent, we received 20 complete responses, 5 incomplete responses, and 10 non-responses over the period of 45 days. Moreover, the response proportion was equally distributed among broadcast and unicast approach. There were 9 questions in all, including both open-ended and closed-ended.

Let us highlight some key comments and observations from some selected Apache reviewers on their experts' opinions.

- **Q:** What motivated the switch from broadcast to unicast? **A:** 12 of the reviewers said that the broadcast were good for discussion (functional/design/customer related announcements/release etc.). However, when it comes to reviewing code, tracking tasks/issues – such as what release of the product they are targeted for, whether an issue has been fixed (and if so, which release or releases, etc.), whom an issue is assigned to, who reported the issue, etc.. – the unicast system is much convenient for reviewers than broadcast, since it reports all (meta)data about a particular patch on one web page.

  > Unicast makes it much easier for the reviewer to leave comments on specific blocks or lines of code, and to visualize the exact changes made by a diff for example. Also, unicast has less traffic (volume of patches circulating) among the reviewers.

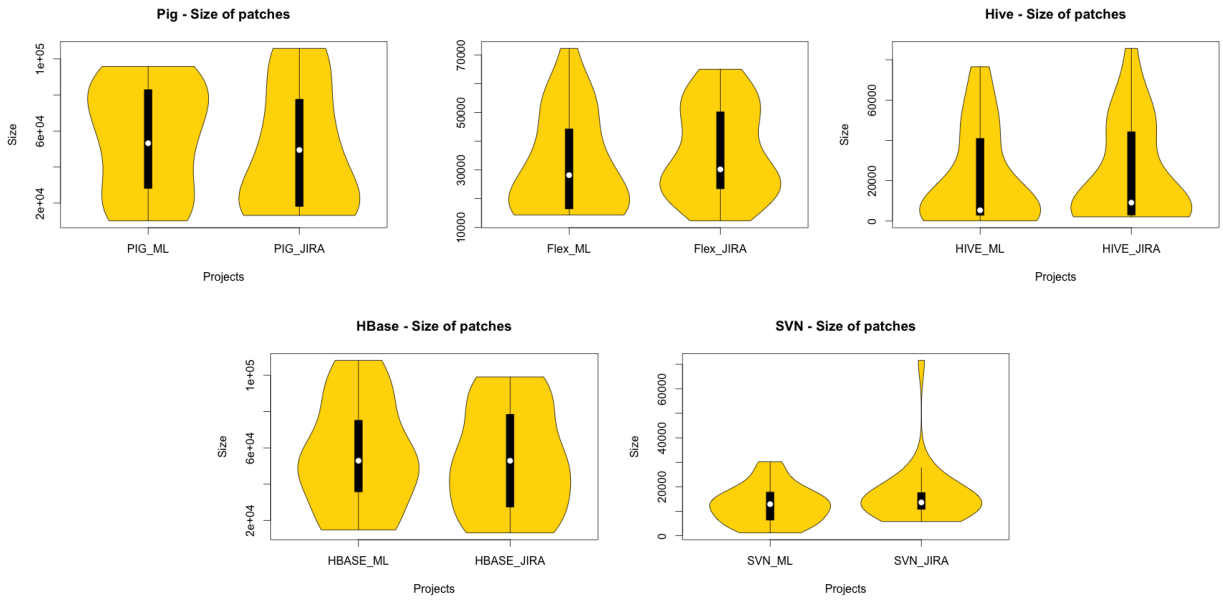


Figure 8: Size of patches.

- **Q:** How hard was it for reviewers to change their review process from broadcast to unicast?
  A: The reviewers unanimously agreed on the switched. Furthermore, they said it was pretty easy learning and using unicast without any interruption to their task.
- **Q:** Based on their personal experiences with both reviewing technologies, what was their main advantages or disadvantages?
  A: The reviewers answered that, unicast technology makes it easier to review patches, track progress on bugs/issues, look up details on old issues, easier to make release notes on what has been fixed, and easier to organize releases.
  However, since it has these many feature, sometimes it can take longer to review patches as the reviewer might want to explore the many options while reviewing patches. Whereas with the broadcast, reviewing patches keeps you focus on the patches and discussions around the patches. On the other hand, due to the broadcasting nature of broadcast technology, every reviewer see the patch, which facilitates the learning process for new developers who join a team (speeding them up quickly), as it animates discussions around code structure, style, and architecture.

> New developers learn about the code structure faster with broadcast than using unicast. The traffic of patches circulating on broadcast is high, because it circulates among all those who are subscribed to the broadcast medium.

IV. Discussion

In this section we discuss our results in more details. We also comment on how our findings agree with Apache reviewers experience using both the broadcast and unicast technologies. First, we comment on the three categories, that is: *Effort*, *Effectiveness* and *Efficiency*.

For RQ1, We address Effort. We measure efforts based on the metrics mentioned in Table I for both the broadcasts and unicast. The statistical test results for the number of reviewers (NR) (see Table II) were a bit expected since we selected projects that were first using broadcasts and then switched to unicast; it is no surprise that the number of reviewers remained almost the same after the transition from broadcasts and unicast. This is also an interesting point to note for this study, we fixed this metric (NR) as a constant and study the effect of the number of revisions (NV) and the review queue (RQu). These two metrics show significant differences between the broadcast and unicast technologies. We did not considered the learning curve for developers, that is the time to learn the new tool during the switch from broadcasts to unicast. We also didn't considered the cultural change. Nevertheless, we analyzed several comments but found none that expresses any such difficulties or challenges. Therefore, we conclude that based on effort, unicast seems to be a better tool for Review than broadcasts.

For RQ2, we address Effectiveness. From the results of the proportion of reviewed patches, which experienced a future bug and the median review rate (see Table II and Figure 7), we observe that patches reviewed through broadcasts are more likely to experience a post release bug than those reviewed on unicast. This is probably due to the good traceability features of unicast technologies such as JIRA. Therefore, Software or-

ganizations should consider adopting mature Review platforms like JIRA, since they are likely to improve the effectiveness of Review activities.

For RQ3, we address Efficiency. Even though the review length seems shorter on broadcasts in comparison to unicast, the difference is not statistically significant. In the case of response delay, the only systems for which we found a statistically significant difference are PIG and HBase. However, these two projects had the largest amount of patches among our studied projects, see Figure 8, which may explain why we observed a statistically significant difference. Typically, we see that the only difference between the broadcast and unicast technology is how reviewers get patches.

Going back to the selection of the projects for this study, all five projects were at their maturity state, therefore we can't attribute any threat to validity to this aspect. Moreover, the size of patches reviewed under the broadcast and the size of patches reviewed using unicast are statistically the same. *Therefore, we can't claim that the results were affected by either the project state or any difference in size*[3].

Analyzing Table II, we observe that the p-values calculated for the seven variables are statistically significant for five metrics (RL, NV, RQs, MRR, and RD).

Moreover, this findings are useful for our analysis because, just as we mentioned on *size* above, the number of reviewers is another variable we should pay close attention to. For example, if the number of reviewers on broadcast had a statistically significant difference with those on unicast, then it could turn out that, the result of our findings were influenced by the numbers of reviewers. That is, we could have concluded that maybe there were more reviewers on the broadcast that is why it took less among of time to review codes than unicast. Whereas, it is not the case in this study. However, since this is not the case, the result of our findings are very likely solely due to the medium on which code review is performed.

Furthermore, let us revisit the survival curve in Figure 6, we notice large steps in the staircase, which suggest that the shape of the survival curves are strongly affected by minor changes in the data, i.e., one reviewer performing an additional review.

Notwithstanding, since the numbers of reviewers are not (statistically) significantly different, we observe large step in the staircases in both the broadcast and unicast. Also, in Table IV we see that unicast reviewers on average, survive more than reviewers on broadcast. This confirms our finding that review done on unicast undergo more rounds than those done on broadcast.

Additionally, the more a patch undergo rounds of reviews, the less likely it can be bug prone, this explains why applying the SZZ algorithm on both medium shows unicast to be more effective than broadcast.

Based on our findings and the Apache reviewers experts' opinion, switching from broadcast to unicast was necessary, because most of the reviewers found unicast easier to use and more useful for code review.

[3]Size of patches for the projects comparing size using broadcast against size using unicast, see Figure 8.

Also, changing from broadcast to unicast, we thought could have caused delay for the reviewers (spending additional time to learn unicast), because we were expecting that the learning curve could have influenced the reviewing process.

Overall, our findings are consistent with the responses given by Apache's reviewers. In RQ3, our results suggest that, unicast technology is more effective in catching bugs than broadcast technology and reviewers advocate that, unicast makes it easier for them to leave comments and visualize the exact changes that have been performed. Two aspects that are important during a bug fixing process. Reviewers also mentioned that the traffic on unicast technologies, which is lower in comparison to the traffic on broadcast technologies; resulting in less distractions during the bug fixing process.

We captured a conversation on the broadcast technology of HBase (among some reviewers) centered on the transition from broadcast to unicast. In this particular conversation, the reviewer was asking in the forum if the switch has already taken place. At that point we could not tell if unicast would be better than broadcast. However, after our analysis and feedback from selected reviewers, we now conclude that it was a justified anxiousness.

From the results of this study, we can comment that Apache's decision to switch to unicast for code review was good and justified.

V. Related Work

Several research works have been carried out in the past on code review, in this section, we present the main contributions that are relevant to this paper. We identify three main areas of contribution that are closely related to our work.

Code review quality.
Kononenko et al. investigated the quality of code review on Mozilla OSS project [28]. They argue that in practice, the process of executing code review can still allow bugs in to the code database or repository, their result confirms this fact with a 54% of reviewed code that were found to be buggy [28]. This also agrees with our result on broadcasts as compared to unicast. To investigate the quality of code review, they used two metrics that we used in our study, the proportion of reviewed patches that experienced a bug and reviewers participation.

Moreover, Morales et al. studied the impact of code review on design quality [39]. Using three projects: Qt, VTKK and ITK, they focused more on anti-pattern, and from their findings, they also suggest that strong code review participation positively impacts software design quality.

Code review practice.
Belli and Crisan [40], proposed an approach to improve the efficiency of code review activities. To evaluate their proposed approach, the authors subdivided reviewers into two independent groups. Each group analyzing the same block of code using two different techniques; one group using their proposed approach and the other group using the traditional manual approach. Reviewers using their proposed approach performed shorter review sessions in comparison to those following the traditional manual technique.

Bavota et al. show that there is a relationship between code review quality and code review practice [29]. Their results reveal that codes that are reviewed before committed reduces the likelihood of inducing bug fixes by 50%. Moreover, they also show that the readability of codes that are committed after being reviewed is significantly higher than the readability of unreviewed codes. They studied the usefulness of the numbers of reviewers as a factor to improve the quality of a code review. In our study, we also computed this metric and obtained similar results that confirm their work. Moreover, their work is based on three open source projects that use the Gerrit code review platform, unlike our studied projects which uses unicast and broadcast technologies. They also use the SZZ [33] algorithm and their result confirms our findings on the role of review media on effectiveness.

A more related work is the empirical study by Bettenburg et al. [23] who studied developers contributions to two systems Android (Profit) and Linux Kernel (non-profit) systems. This work inspired our approach of using the broadcast and unicast technologies to study tool-based code review [16].

Code review medium.
De Alwis et al. highlight the importance of changing version control systems, in this context they show that more and more open and closed source projects are switching from centralized version control systems (CVCS) to decentralized version control systems (DVCS) [41]. In their work, they points out the main differences between CVCS and DVCS, and explained the rationales behind the transition and some key benefits for projects; giving reasons for the transition. They also capture some interesting discussions among developers discussing about the transition, in our work, we also captured some reviewers discussion on the transition from broadcast to unicast. Moreover, they points out major weaknesses of the CVCS and show how DVCS will benefit non-committers. Also, they identify the following projects, which are switching to DVCS: (Perl, OpenOffice, NetBSD, Python. However, their work was limited to what projects team members believe to be the impact of switching to a DVCS.

VI. Threats to Validity

In this section, we discuss the limitations of our work following common guidelines for empirical studies [42].

A. Construct validity threats

Relate to the meaningfulness of our measurement results, in other words, this threat is solely due to errors in the measurement of metrics. To compute our metrics, we used Mailminer2 [26] that converts the *.mbox* files into postgressql relational Database records, email threads reconstruction can be problematic, for example some emails may contain special characters that causes some of the threads not to be converted to the database record; the risk for data loss is much larger in broadcast than unicast. Furthermore, unicast dataset is more recent than the broadcast dataset, as a consequence, any differences observed could be due to cultural changes in the projects rather than the Review infrastructure used. Additionally, since different kinds of projects may have different complexity, it may be a better idea to use different kinds of projects with varying complexity to compare these two review technologies.

B. Threats to internal validity

Relate to alternative explanations to describe our findings. The implementation of the SZZ used can be a threat to this study, moreover, our design window can also be a potential threat because not all the projects switched at the same time. That is, inappropriate choice of the time window sizes might have lead to ecological inference fallacy [43]. The choice of metrics can be a threat as well to this study. However, we selected well known metrics from the code review literature to mitigate this threat. Also, for a trade-off between either comparing broadcast and unicast in different systems at the same time (no pre/post issue, but so many other confounding factors and noise), or comparing in the same system before and after (closest to have minimal confounding factors, although not perfect.) We selected old projects, which used the broadcast for reviewing patches and then switched from broadcast to unicast.

C. Conclusion validity threats

Concerns the relation between the treatment and its outcome. To address this, we were attentive not to violate the assumptions of our null hypothesis. We claim correlation and not causality.

D. Reliability validity threats

Replication is important in science, this treat addresses how this work can be replicated. To this effect, we attempt to provide necessary details to replicate our study. The repositories and tools we used in this study are open source.

VII. Conclusion

In this paper, we empirically studied review data of five Apache projects that switched from broadcasts code review to unicast code review environment; a web-based tool. Our objective was to understand the impact of review technology on review effectiveness and quality. Our results suggest that broadcasts reviews are twice faster than unicast reviews. However, unicast's review quality outperforms that of broadcast. Additionally, using the SZZ algorithm to track reviewed patches that experienced future bugs, we observed that patches reviewed through broadcasts are more likely to experience a post release bug than those reviewed on unicast.

Our findings suggest that the medium used for code review can affect the effectiveness and hence the quality of review activities. Software organizations should consider adopting mature Review platforms like unicast platforms, since they are likely to improve the effectiveness of Review activities.

References

[1] P. Thongtanunam, S. McIntosh, A. E. Hassan, and H. Iida, "Investigating code review practices in defective files: An empirical study of the qt system," in *Mining Software Repositories (MSR), 2015 IEEE/ACM 12th Working Conference on*, May 2015, pp. 168–179.

[2] Y. Wang, X. Zhang, L. Yu, and H. Huang, "Quality assurance of peer code review process: A web-based mis," in *Computer Science and Software Engineering, 2008 International Conference on*, vol. 2, Dec 2008, pp. 631–634.

[3] Y. Tymchuk, "Treating software quality as a first-class entity," in *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on*, Sept 2015, pp. 594–597.

[4] M. E. Fagan, "Design and code inspections to reduce errors in program development," *IBM Systems Journal*, vol. 15, no. 3, pp. 182–211, 1976.

[5] M. Beller, A. Bacchelli, A. Zaidman, and E. Juergens, "Modern code reviews in open-source projects: Which problems do they fix?" in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014, 2014, pp. 202–211.

[6] B. Meyer, "Design and code reviews in the age of the internet," *Commun. ACM*, vol. 51, no. 9, pp. 66–71, Sep. 2008.

[7] A. Bosu, M. Greiler, and C. Bird, "Characteristics of useful code reviews: An empirical study at microsoft," in *Proceedings of the 12th Working Conference on Mining Software Repositories*, ser. MSR '15, 2015, pp. 146–156.

[8] V. Balachandran, "Fix-it: An extensible code auto-fix component in review bot," in *Source Code Analysis and Manipulation (SCAM), 2013 IEEE 13th International Working Conference on*, Sept 2013, pp. 167–172.

[9] V. Balachandran, "Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13, 2013, pp. 931–940.

[10] Y. Jiang, B. Adams, F. Khomh, and D. M. German, "Tracing back the history of commits in low-tech reviewing environments: A case study of the linux kernel," in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '14. New York, NY, USA: ACM, 2014, pp. 51:1–51:10. [Online]. Available: http://doi.acm.org/10.1145/2652524.2652542

[11] N. Bettenburg, E. Shihab, and A. E. Hassan, "An empirical study on the risks of using off-the-shelf techniques for processing mailing list data," in *ICSM'09: Proceedings of the 25th IEEE International Conference on Software Maintenance*. IEEE Computer Society, 2009, pp. 539–542.

[12] P. Thongtanunam, X. Yang, N. Yoshida, R. G. Kula, A. E. C. Cruz, K. Fujiwara, and H. Iida, "Reda: A web-based visualization tool for analyzing modern code review dataset," in *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*, Sept 2014, pp. 605–608.

[13] A. Guzzi, A. Bacchelli, M. Lanza, M. Pinzger, and A. v. Deursen, "Communication in open source software development mailing lists," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, ser. MSR '13, 2013, pp. 277–286.

[14] P. C. Rigby and A. E. Hassan, "What can oss mailing lists tell us? a preliminary psychometric text analysis of the apache developer mailing list," in *Proceedings of the Fourth International Workshop on Mining Software Repositories*, ser. MSR '07, 2007, pp. 23–. [Online]. Available: http://dx.doi.org/10.1109/MSR.2007.35

[15] P. Rigby, B. Cleary, F. Painchaud, M. A. Storey, and D. German, "Contemporary peer review in action: Lessons from open source development," *IEEE Software*, vol. 29, no. 6, pp. 56–61, Nov 2012.

[16] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13, 2013, pp. 712–721.

[17] M. Squire, "Should we move to stack overflow? measuring the utility of social media for developer support," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 2, May 2015, pp. 219–228.

[18] V. J. Hellendoorn, P. T. Devanbu, and A. Bacchelli, "Will they like this?: Evaluating code contributions with language models," in *Proceedings of the 12th Working Conference on Mining Software Repositories*, ser. MSR '15, 2015, pp. 157–167.

[19] C. Williams and J. Spacco, "Szz revisited: verifying when changes induce fixes," in *Proceedings of the 2008 workshop on Defects in large software systems*. ACM, 2008, pp. 32–36.

[20] "Replication:guide to our tools and dataset," https://bitbucket.org/foundjem/ml-issuetk/src, accessed: 2016-03-17.

[21] P. C. Rigby and C. Bird, "Convergent contemporary software peer review practices," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2013, 2013, pp. 202–212.

[22] 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*. ACM, 2008, pp. 541–550.

[23] N. Bettenburg, A. E. Hassan, B. Adams, and D. M. German, "Management of community contributions," *Empirical Software Engineering*, vol. 20, no. 1, pp. 252–289, 2013. [Online]. Available: http://dx.doi.org/10.1007/s10664-013-9284-6

[24] "Apache-Software-Foundation developer mailing list," http://mail-archives.apache.org/mod_mbox/hbase-dev/, accessed: 2016-03-17.

[25] "Apache-Software-Foundation jira issue tracking system," https://issues.apache.org/jira/secure/BrowseProjects.jspa#all, accessed: 2016-03-17.

[26] N. Bettenburg, E. Shihab, and A. E. Hassan, "An empirical study on the risks of using off-the-shelf techniques for processing mailing list data," in *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*, Sept 2009, pp. 539–542.

[27] "Apache-Software-Foundation github mirrior site," https://github.com/apache, accessed: 2016-03-17.

[28] O. Kononenko, O. Baysal, L. Guerrouj, Y. Cao, and M. W. Godfrey, "Investigating code review quality: Do people and participation matter?" in *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on*, Sept 2015, pp. 111–120.

[29] G. Bavota and B. Russo, "Four eyes are better than two: On the impact of code reviews on software quality," in *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on*, Sept 2015, pp. 81–90.

[30] C. F. Kemerer and M. C. Paulk, "The impact of design and code reviews on software quality: An empirical study based on psp data," *IEEE Trans. Softw. Eng.*, vol. 35, no. 4, pp. 534–550, Jul. 2009.

[31] "Apache-Software-Foundation developer mailing list," https://smartbear.com/learn/code-review/best-practices-for-peer-code-review/, accessed: 2016-03-17.

[32] D. B. Bisant and J. R. Lyle, "A two-person inspection method to improve prog ramming productivity," *IEEE Transactions on Software Engineering*, vol. 15, no. 10, pp. 1294–1304, Oct 1989.

[33] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, "Extracting structural information from bug reports," in *Proceedings of the 2008 international working conference on Mining software repositories*. ACM, 2008, pp. 27–30.

[34] A. Meneely, H. Srinivasan, A. Musa, A. Rodriguez Tejeda, M. Mokary, and B. Spates, "When a patch goes bad: Exploring the properties of vulnerability-contributing commits," in *Empirical Software Engineering and Measurement, 2013 ACM/IEEE International Symposium on*. IEEE, 2013, pp. 65–74.

[35] G. Macbeth, E. Razumiejczyk, and R. D. Ledesma, "Cliff's delta calculator: A non-parametric effect size program for two groups of observations," *Universitas Psychologica*, vol. 10, no. 2, pp. 545–555, 2011.

[36] M. Mukaka, "A guide to appropriate use of correlation coefficient in medical research," *Malawi Medical Journal*, vol. 24, no. 3, pp. 69–71, 2012.

[37] O. Baysal, O. Kononen, R. Holmes, and M. W. Godfrey, "The influence of non-technical factors on code review," in *2013 20th Working Conference on Reverse Engineering (WCRE)*. IEEE, 2013, pp. 122–131.

[38] B. D. Sethanandha, "Improving open source software patch contribution process: methods and tools," in *Software Engineering (ICSE), 2011 33rd International Conference on*, May 2011, pp. 1134–1135.

[39] R. Morales, S. McIntosh, and F. Khomh, "Do code review practices impact design quality? a case study of the qt, vtk, and itk projects," in *Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on*, March 2015, pp. 171–180.

[40] F. Belli and R. Crisan, "Empirical performance analysis of computer-supported code-reviews," in *Proceedings The Eighth International Symposium on Software Reliability Engineering*, Nov 1997, pp. 245–255.

[41] B. de Alwis and J. Sillito, "Why are software projects moving from centralized to decentralized version control systems?" in *Cooperative and Human Aspects on Software Engineering, 2009. CHASE '09. ICSE Workshop on*, May 2009, pp. 36–39.

[42] R. K. Yin, *Case study research: Design and methods*. Sage publications, 2013.

[43] D. Posnett, V. Filkov, and P. Devanbu, "Ecological inference in empirical software engineering," in *Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on*, Nov 2011, pp. 362–371.