Writing Acceptable Patches: An Empirical Study of Open Source Project Patches

Yida Tao*, DongGyun Han† and Sunghun Kim*
*Department of Computer Science and Engineering
The Hong Kong University of Science and Technology
{idagoo, hunkim}@cse.ust.hk
†KAIST Institute for IT Convergence
Korea Advanced Institute of Science and Technology
handk@kaist.ac.kr

***Abstract*—Software developers submit patches to handle tens or even hundreds of bugs reported daily. However, not all submitted patches can be directly integrated into the codebase, since they might not pass patch review that is adopted in most software projects. As the result of patch review, incoming patches can be rejected or asked for resubmission after improvement. Both scenarios interrupt the workflow of patch writers and reviewers, increase their workload, and potentially delay the general development process.**

In this paper, we aim to help developers write acceptable patches to avoid patch rejection and resubmission. To this end, we derive a comprehensive list of patch rejection reasons from a manual inspection of 300 rejected Eclipse and Mozilla patches, a large-scale online survey of Eclipse and Mozilla developers, and the literature. We also investigate which patch-rejection reasons are more decisive and which are difficult to judge from the perspective of patch reviewers. Our findings include 1) suboptimal solution and incomplete fix are the most frequent patch-rejection reasons 2) whether a patch introduces new bugs is very important yet very difficult to judge 3) reviewers reject a large patch not solely because of its size, but mainly because of the underlying reasons that induce its large size, such as the involvement of unnecessary changes 4) reviewers consider certain problems to be much more destructive than patch writers expect, such as the inconsistency of documentation in a patch and 5) bad timing of patch submission and a lack of communication with team members can also result in a negative patch review.

I. Introduction

Bugs are inevitable in the process of software development. After a bug is found and reported, developers are assigned to write its patch, which is then reviewed by patch reviewers (Figure 1). Given tens or even hundreds of bugs reported daily for individual projects [1] and limited development resources, an ideal scenario for a patch is to be accepted immediately after its first review.

In practice, however, a patch often iterates the "review-reject-resubmit" process (Figure 1) for quite a long time. For instance, nearly 40% of the patches for the Mozilla Firefox project are resubmitted at least once [2]. A patch-resubmission iteration can take days or even months before the patch is finally accepted [1][2][3]. There are also a non-negligible quantity of patches that never make their way to the final acceptance stage. Rigby and German reported that 56% of patches were rejected in the Apache HTTP server project [4].

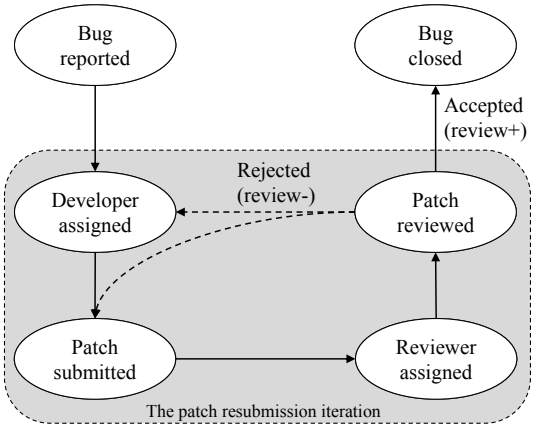

Fig. 1: A typical patch review process. The central loop indicates that a patch is rejected and resubmitted for review.

Weißgerber et al. reported that about 60% of patches were rejected in the FLAC and OpenAFS projects [5].

Patch rejection potentially interrupts the workflow of both patch writers and reviewers, who are forced to repeatedly revisit patches. Consequently, the general development process can be delayed or even frozen. To avoid such situations, it is desirable for developers to have their patches accepted right after the very first review.

In this paper, we investigate the problem of *how to write acceptable patches*, which is decomposed into three research questions as shown below. We start with learning from "past failures", that is, why patches are rejected (RQ1). Figure 2 shows six main reasons for rejecting a patch that has been submitted to resolve the Eclipse bug #197448. Note that for a given patch, its rejection reasons may not be equally deterministic. For example, *causing a compilation error* in Figure 2 seems to be the primary reason for rejection. Hence, in RQ2, we investigate which reasons are more important for patch reviewers to make decisions. Based on these results, we finally propose guidelines to help developers write acceptable patches (RQ3).

- **RQ1**: Why are patches rejected?
- **RQ2**: Which reasons are more decisive when patch reviewers reject a patch?

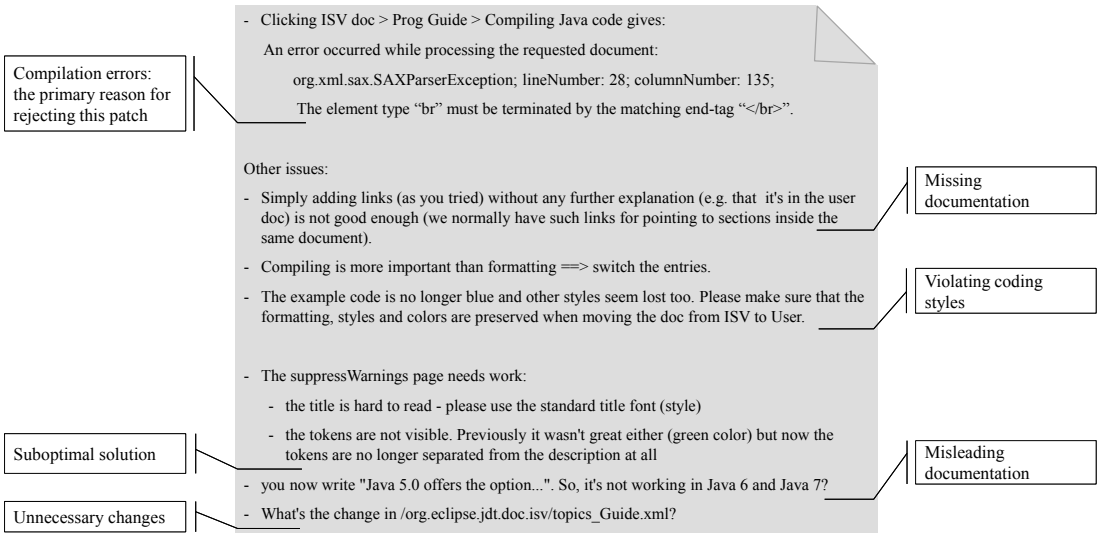

Fig. 2: Review comments of a patch for Eclipse bug #197448. Its reviewer pointed out six problems with this patch, of which the *compilation error* is the primary "culprit" for this patch rejection.

- **RQ3**: What are the guidelines for writing acceptable patches?

Our study design consists of three parts. First, we extracted 300 rejected patches from the development history of Eclipse and Mozilla, and manually inspected their patch review comments to understand why they were rejected. Second, we conducted a large-scale online survey involving 246 Eclipse and Mozilla developers. From this online survey, we mainly investigate reasons that are the most decisive for rejecting a patch. We further surveyed the past literature on patch review to complement the empirical results of our manual inspection and developer survey.

We collectively identify 30 reasons for patch rejection, which fall into five major categories: *problematic implementation or solution*, *difficult to read or maintain*, *deviating from the project focus or scope*, *affecting the development schedule*, and *lack of communication or trust*. We find that although implementation and maintainability are the primary concerns in a patch review, the timing of a patch submission (e.g., whether a patch is submitted when the release is approaching) and the communication between patch writers and reviewers also affect patch review outcomes.

For a more detailed classification of the reasons for patch rejection, we observe that patches are most frequently rejected for being *suboptimal* or *incomplete*, both of which are also very decisive. On the other hand, although *introducing new bugs* is highly decisive for patch rejection, developers consider it to be very difficult to judge. Our manual inspection consistently reveals less occurrence of patches that are rejected for this particular reason. We also find that patch size is in fact a minor factor. Instead, reviewers tend to judge the underlying reasons that cause a large patch size, for example whether a patch includes unnecessary changes.

Our study also reveals the disconnection between patch writers and reviewers in terms of their criteria of patch quality. For instance, reviewers considered a patch having *inconsistent or misleading documentation* to be highly unacceptable. However, patch writers assigned significantly lower decisive scores to this particular reason. Interestingly, for all the patch-rejection reasons that received significantly different decisive scores from patch reviewers and writers, reviewers consistently assigned higher scores than patch writers. This result suggests that patch reviewers consider certain issues of a patch more destructive than patch writers expect.

Overall, our paper makes the following contributions:

- **A comprehensive list of reasons for patch rejection**: from our manual patch inspection, online developer survey and literature survey, we collectively identify 30 detailed patch-rejection reasons that fall into five major categories.
- **An in-depth empirical investigation of patch-rejection reasons**: from our survey of 246 developers, we investigate how decisive each reason for patch rejection is and how difficult it is for patch reviewers to judge. We also explore potential disconnections between patch writers and reviewers in terms of their patch evaluation criteria.
- **Guidelines for writing acceptable patches**: according to our investigation on rejected patches, we propose actionable guidelines to help developers write acceptable patches.

The remainder of this paper is organized as follows. Section II introduces our study design. Section III reports our study results with respect to the first two research questions, followed by a discussion in Section IV. Section V answers RQ3 by proposing guidelines for writing acceptable patches. Section VI discusses the threats to validity of our study, followed by a survey of related work in Section VII. Section VIII concludes the paper.

Patch attachments for Eclipse bug#233156 (Reported on 2008-05-21)	
~~spell checking performance test~~ (4.39 KB, patch) 2009-11-23	*no flags*
~~performance test~~ (5.69 KB, patch) 2009-11-30	*no flags*
~~performance test + fix~~ (12.95 KB, patch) 2009-12-01	review-
~~fix~~ (7.63 KB, patch) 2010-01-04	review-
performance test (6.53KB, patch) 2010-01-04	review+
fix (10.08 KB, patch) 2010-01-14	review+

Fig. 3: To fix Eclipse bug #233156, five patches were submitted before the sixth patch was finally accepted two years after the bug was reported.

II. Study Design

In this section, we introduce our study design, which consists of three parts: a manual inspection of real OSS patches (Section II-A), an online survey of developers (Section II-B) and a literature survey (Section II-C).

A. Manual Patch Inspection

To understand why patches are rejected (RQ1), we manually inspected rejected patches and the associated review comments from two open source projects, Eclipse and Mozilla. Both projects use Bugzilla[1] to track bug reports, patch submissions and patch reviews.

In Bugzilla, a patch with the *review+* flag is typically considered acceptable by patch reviewers while a patch with the *review-* flag fails the review and needs to be revised [2][6]. For our study, we extracted only the explicitly rejected patches with *review-* flags and review comments. Since reviewers typically explain the problems with a rejected patch in their review comments (Figure 2), we were able to reasonably deduce why it is rejected. We ignored patches with *no flags* since they may have been implicitly rejected or accepted (Figure 3).

Accordingly, we extracted 238 explicitly rejected patches from Eclipse and 673 from Mozilla. From these patches, we randomly selected 300 for manual inspection (Table I). Two of the authors individually inspected these 300 rejected patches and particularly their review comments to identify their rejection reasons. Then, the results were compared and similar reasons were merged to one succinct term (e.g., "the patch fails a test" and "test fail" were merged to "test failures"). After this step, the two evaluators agreed on 4/5 patch-rejection reasons, reaching a Cohen's Kappa of 0.75 that is considered to be "good agreement" [7]. For the remaining 1/5 disagreed reasons, we invited an external computer science postgraduate to the discussion and finally reached an agreement. In total, we manually identified 12 reasons for patch rejection, which are discussed in Section III-A.

[1] http://www.bugzilla.org/

TABLE I: The number of explicitly rejected patches within the studied period. From these patches, we randomly selected 300 for manual inspection.

Project	**Period**	**Patches w/ *review-* flag**	**Manually inspected**
Eclipse	2001.10 - 2011.08	238	162
Mozilla	2011.01 - 2011.08	673	138
		Total	300

TABLE II: The number of developers who were invited to our survey and the number of those who completed the survey. The survey response rate is shown in the parenthesis.

	Patch Writer	**Patch Reviewer**	**Total**
Eclipse	25 / 233	48 / 246	73 / 479 (15.2%)
Mozilla	106 / 952	67 / 354	173 /1306 (13.2%)
Total	131 /1185 (11.1%)	115 / 600 (19.2%)	246 /1785 (13.8%)

B. Developer Survey

Based on our manual inspection results, we conducted an online survey with Eclipse and Mozilla developers. In this survey, we asked developers to rate how decisive these 12 reasons for patch rejection are on a 5-point Likert scale (RQ2). To complement our manual inspection, we also asked developers to supplement additional patch-rejection reasons that they considered important but were not listed in our survey. Furthermore, as an attempt to quantify the cost of patch review, we asked developers to rate the difficulty of judging each reason for patch rejection on a 5-point Likert scale (Section IV-B).

We explicitly distinguish patch writers and reviewers from our survey respondents since these two roles might hold different opinions on patch quality (Section IV-A). Specifically, we consider a developer who has written at least one patch as a writer and who has reviewed at least one patch as a reviewer. If a developer meets both criteria, we consider him/her as a reviewer.

Table II shows our survey statistics. We identified 233 patch writers and 246 reviewers from the Eclipse project and invited them to our online survey via email. Among them, 25 writers (10.73%) and 48 reviewers (19.51%) completed the survey. For the Mozilla project, we identified 952 writers and 354 reviewers. Among them, 106 writers (11.13%) and 67 reviewers (18.92%) completed the survey.

In total, we received 246 responses, of which 131 were from patch writers and 115 were from patch reviewers. As shown in Table II, we have reached a survey response rate of 13.8% (246/1785), which is comparable to that of similar studies [8][9].

C. Literature Survey

To complement the empirical results of our manual inspection and developer survey, we further surveyed the past literature related to patch review. We focused on work that has

TABLE III: Patch-rejection reasons identified from our manual inspection. The second column shows the example of patch review comments, whose project name and bug ID are shown in the last column.

Patch-rejection reason	Example of patch review comments	Project-BugID
Compilation errors	I will not review a patch that causes errors in my workspace. As said before: make sure you have API tools enabled and a R3.5 baseline set.	Eclipse-78522
Test failures	The provided patch causes about 20 tests to fail. Either the change really breaks something, or it has side-effects that need the tests to be changed, that means that it changes the expected behavior of the generator.	Eclipse-331875
Introducing new bugs	The patch fixes the CCE but introduces a new bug: the returned key string is wrong in the normal case i.e., it includes the “”at the end.	Eclipse-247012
Inconsistent or misleading documentation	The note is unclear. “As per ...” sounds like we follow the spec. But since we don’t, this should be stated explicitly (“Note: This deviates from JLS3 14.3...”). Furthermore, it’s confusing that you use differing terms “anonymous type” and “anonymous inner classes” for the same thing.	Eclipse-339337
Suboptimal solution	I honestly don’t want all this complexity for this user pref . . . Much easier will be to add a link from the Email Preferences tab pointing to email-related user prefs once bug 589138 is implemented	Mozilla-589128
Duplication	What I now don’t like is that we have two methods which almost do the same thing but have different names: *#packageChanged()* and *#getPackageStatus(packName)*.	Eclipse-393161
Including unnecessary changes	Removed this unnecessary check from *#getNextElseOffset: if(then == null) return -1;*	Eclipse-377141
Incomplete fix	I couldn’t test this patch, as it seems to be missing the change to browser.inc that adds secondaryToolbarButtons.	Mozilla-877335
Violating coding style guidelines	Per our Bugzilla guideline, we never leave *|if (|* alone on its own line.	Mozilla-637981
Bad naming	*JavaCompareUtilities.getActiveEditor(IEditorPart)* has wrong name as it simply works with the given part (doesn’t matter if active or not)	Eclipse-260531
Missing documentation	In the OverviewRuler class Javadoc I would mention that it uses non-saturated colors unless *setUseSaturatedColorPreference(...)* gets called.	Eclipse-341808
Patch size too large	Here is a patch smaller than 250 line.	Eclipse-344125

been published in the past decade (i.e., 2004–2014) to ensure the freshness of our survey results. Our literature survey mainly consists of *discovering*, *expanding*, and *filtering* steps.

We first discovered patch-review related work by searching Google Scholar [2] with the keywords “patch review” and “patch acceptance” [3], which rendered an initial set of publications. We then expanded these publications by searching their references and citations. We iterated this expanding process until no new publication emerged. During these two steps, we read only the title and abstract of a search result to quickly determine whether it was related to patch review. At this point, we identified 26 related papers, which were published in 14 different venues including ESEC/FSE (the European Software Engineering Conference/Symposium on the Foundations of Software Engineering), ICSE (the International Conference on Software Engineering), WCRE (the Working Conference on Reverse Engineering) and MSR (the Working Conference on Mining Software Repositories).

In the filtering step, we read these 26 papers carefully. We found that although these studies fall into the field of patch review, their focuses are quite diverse. For example, a number of studies focus on understanding and characterizing the general patch review process [10][11][12][13][14]. Other studies have investigated the reviewers involved, the defects uncovered, and the tools used in the patch review [15][16][17]. Another line of work predicts patch review outcome or patch acceptance time using features such as patch size and reviewer experience [18][19]. A few studies have investigated the correlation between patch acceptance and patch size or patch-writers’ expertise [5][2][19][20], However, these analyses are mainly post-mortem and less likely to reflect reviewers’ initial intention of rejecting patches.

Surprisingly, out of these 26 papers, only 3 explicitly discuss reasons for patch acceptance or rejection. Rigby and Storey summarized six technical reasons and six non-technical reasons (e.g., project politics) why patches are rejected [12]. Nurolahzade et al. presented five types of reviewer feedback: *implementation*, *functionality and usability*, *documentation*, *coding standards*, and *performance* [21]. Rigby et al. examined the Github pull request and manually classified reasons why pull requests are rejected. They found that 27% of the rejected pull requests are due to concurrent modifications of the code in project branches, 16% are due to patch contributors’ misunderstanding of the project direction, 13% are due to implementation errors and 10% are due to project process and quality requirements [22].

We combined the patch-rejection reasons extracted from the literature with our empirical results. In total, we identi-

[2]http://scholar.google.com

[3]We also tried other keywords such as “code review” and “patch reject”, which, however, rendered mostly irrelevant results.

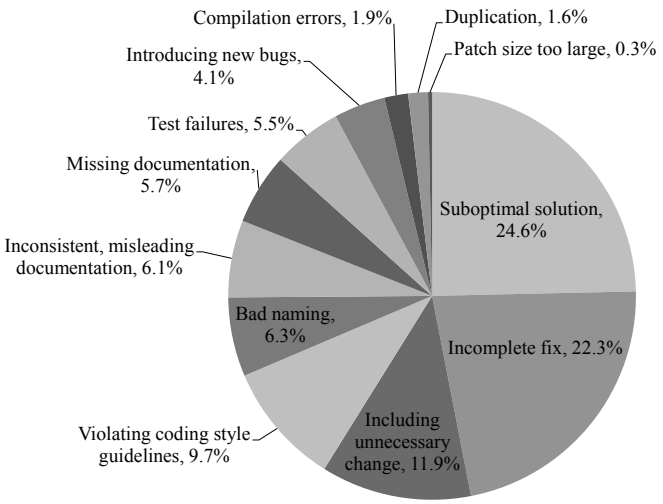

Fig. 4: The percentage of patch-rejection reasons identified in our manual inspection.

fied 30 detailed patch-rejection reasons, which are shown in Table IV and discussed in Section III-A.

III. Results

A. RQ1: Why are Patches Rejected?

Table III lists the patch-rejection reasons identified from our manual inspection of Eclipse and Mozilla patches. Specifically, Table III shows examples of review comments corresponding to each reason. Figure 4 shows the percentage of patch-rejection reasons identified from our manual inspection. "Suboptimal solution" and "incomplete fix" turn out to be top reasons for patch rejection. We revisit this finding in Section IV-B.

As introduced in Section II, our manual inspection results are further complemented by an online developer survey and the past literature. We finally identify 30 detailed patch-rejection reasons that fall into five major categories, as shown in Table IV.

Problematic implementation or solution: a patch can be rejected if it causes compilation errors, fails tests, or introduces new bugs. Also, it can be rejected for not completely fixing the bug (e.g., it misses corner cases). In addition to the implementation correctness and completeness, patch reviewers also consider whether the solution proposed in a patch is the best. As shown in Figure 4, nearly a quarter of the patch-rejection reasons are "suboptimal solution". This indicates that in many cases, reviewers reject a patch not because it does not work, but because there exists a simpler or more efficient way of fixing the bug. In addition, a patch is also likely to receive negative reviews if developers impose their personal preferences on the patch solution, which can be too aggressive for end users.

Difficult to read or maintain: an unreadable patch is likely to be rejected. For example, it may use unclear identifier names or violate coding style guidelines. A more general concern is how a patch affects the project's maintainability. For example,

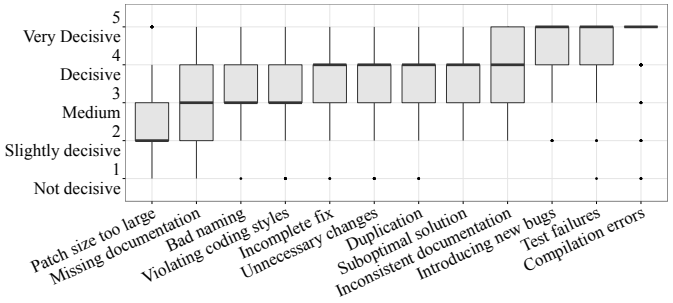

Fig. 5: How decisive each reason is to reject a patch. These scores are collected from all the survey respondents.

patch reviewers tend to respond negatively if a patch includes unnecessary or duplicate changes, or uses deprecated APIs. A patch without proper documentation or test cases is also not favored. In addition, a patch that changes internal APIs or affects many other modules can be rejected for being too risky.

Deviating from the project focus or scope: in addition to implementation details, patch reviewers also evaluate the high-level idea of a patch. If the issue addressed in a patch is irrelevant or out of the primary focus of the project, the patch is likely to be rejected.

Affecting the development schedule: while the prior patch-rejection reasons are closely related to the content of a patch, the timing of a patch submission can also affect its review outcome. A patch can be rejected if it is too late for the current release cycle, or if it freezes the development of other features thus defers the entire development schedule.

Lack of communication or trust: a lack of communication between patch writers and reviewers might also yield to a negative patch review. Our survey respondents explicitly pointed out that the responsiveness of patch writers is very important for patch review. Reviewers also tend to have low confidence in *immature* patches [19], which are submitted without prior discussion with other team members. In addition, reviewers' trust in patch writers in terms of their expertise or reputation also affects their patch review decisions, as pointed out by both our survey respondents and the literature [2][19].

As shown in Table IV, the patch-rejection reasons identified by our manual inspection, developer survey, and the literature are mostly consistent. However, these three approaches also complement each other. For example, Rigby and Storey have observed that a patch can be rejected if it includes large changes yet offers limited improvement over the existing code [12]. This reason for rejecting a patch that has *low cost-benefit* has not been observed from our manual inspection nor the developer survey. On the other hand, our developer survey reveals several patch-rejection reasons, such as *no accompanied test cases*, *changes to internal APIs*, and *submitter unresponsiveness*, that have not been covered in the literature.

B. RQ2: Which Reasons Are More Decisive When Reviewers Reject a Patch?

Table IV shows 30 reasons why a patch can be rejected. However, not all these reasons are equally decisive for rejecting

TABLE IV: Reasons for rejecting a patch. These reasons are jointly derived from the manual inspection of 300 Eclipse and Mozilla rejected patches, a large-scale survey of Eclipse and Mozilla developers, and the past literature.

Category	Detailed reasons	Source		
		Manual Inspection	Developer Survey	Literature Survey
Problematic implementation or solution	Compilation errors	✓		
	Test failures	✓		[22]
	Incomplete fix	✓		[12]
	Introducing new bugs	✓		[12]
	Wrong direction		✓	
	Suboptimal solution	✓		[22]
	Solution too aggressive or hostile for end users		✓	
	Performance		✓	[21]
	Security		✓	
Difficult to read or maintain	Violating coding style guidelines	✓		[12][21]
	Bad naming	✓		
	Patch size too large	✓		[5]
	Missing documentation	✓		[21]
	Inconsistent or misleading documentation	✓		
	No accompanied test cases		✓	
	Integration conflicts with existing code		✓	[12]
	Including unnecessary changes	✓		[12]
	Duplication	✓		[22]
	Misuse of (deprecated) API or library		✓	[12]
	Changes to internal APIs		✓	
	Not well isolated		✓	
	Low cost-benefit			[12]
Deviating from the project focus or scope	Irrelevant or obsolete		✓	[22]
	Not of core interest		✓	[12]
Affecting the development schedule	Freeze the development of other features		✓	[12]
	Low urgency		✓	
	Too late in the release cycle		✓	
Lack of communication or trust	Patch writers are not responsive		✓	
	No discussion prior to the patch submission		✓	
	Patch-writers' expertise and reputation		✓	[2]

a patch. This is confirmed by our developer survey results (Figure 5), from which we make several interesting observations.

"Introducing new bugs" is as decisive as "compilation errors" and "test failures" to reject a patch: as shown in Figure 5, a patch is highly likely to be rejected if it causes compilation errors or fails tests. While this is to some extent expectable, it is interesting to see that reviewers have almost the same low tolerance for patches that introduce new bugs. In such cases, a patch that intends to fix a bug instead worsens the situation by introducing new problems.

Inconsistent or misleading documentation is highly unacceptable and much worse than missing documentation: surprisingly, a patch that contains inconsistent or misleading documentation is also highly unacceptable, as this reason receives the 4th highest decisive score right next to "introducing new bugs" (Figure 5). Specifically, compared to "missing documentation", "inconsistent or misleading documentation" is significantly more decisive in rejecting a patch, with the p-value of the Mann-Whitney U test equals to $0.00 < 0.05$ [23]. We believe that one plausible explanation is that compared to missing documentation, inconsistent documentation might mislead developers thus incur more damage in the long-term development and maintenance of the project.

Large patch size rarely matters, whereas including unnecessary changes in a patch does: although we have observed a few cases in our manual inspection where patches were rejected for changing too many lines, developers in general consider the size of a patch to be a minor factor for patch rejection. In some sense, this finding contradicts prior studies that have emphasized the sole correlation between the size of a patch and its review outcome [5][24]. As one survey respondent pointed out, rather than the patch size itself, the underlying reasons that led to the large patch size is the essential information:

> *The "patch size" criteria is ambiguous. Why is the patch large? If it's necessary, that's not a problem. If it's large because of some other flaw (such as one of the many others enumerated in this survey), then it would be.*

For instance, a patch can be large because it includes unnecessary changes, which might be the real "culprit" for its rejection. According to Figure 5 and the Mann-Whitney U test, "including unnecessary changes" is significantly more decisive than "patch size too large", with a p-value equals to $0.00 < 0.05$.

TABLE V: The Mann-Whitney U test results on the decisive scores received from patch writers and reviewers. A bold p-value (< 0.05) means that the scores are significantly different between these two roles.

Patch-rejection reason	p-value
Compilation errors	0.068
Test failures	**0.000**
Incomplete fix	**0.003**
Introducing new bugs	**0.009**
Violating coding style guidelines	0.681
Bad naming	0.509
Duplication	0.051
Suboptimal solution	**0.011**
Including unnecessary changes	0.226
Inconsistent, misleading documentation	**0.005**
Missing documentation	0.597
Patch size too large	0.135

Figure 5 also shows that "suboptimal solution", "duplication" and "incomplete fix" are in general decisive for rejecting a patch, while "violating coding styles" and "bad naming" are considered to be less decisive.

In addition to the patch-rejection reasons already included in the survey, 44 out of the 246 respondents supplemented additional reasons that they considered important. The topmost mentioned reason was "no accompanied test cases", as 9 respondents considered a patch submitted without test cases to be unacceptable. "Performance" and "patch-writers' expertise and reputation" were both mentioned by 5 respondents.

IV. Discussion

A. Different Views from Patch Writers and Patch Reviewers

As described in Section II-B, the respondents of our online survey have two different roles: patch writers and patch reviewers. In this subsection, we explore whether these two roles are unanimous in their patch evaluation criteria.

For each patch-rejection reason included in our survey, we performed the Mann-Whitney U test on its decisive scores received from patch writers and reviewers. As shown in Table V, these two roles have significantly different decisive scores for five patch-rejection reasons: *test failures*, *incomplete fix*, *introducing new bugs*, *suboptimal solution*, and *inconsistent documentation*. Figure 6, which shows the average decisive scores of these five reasons by patch writers and reviewers, further reveals an interesting trend: *patch reviewers consistently consider these reasons more decisive for rejecting a patch compared to patch writers*.

This result indicates a potential discrepancy between patch writers and reviewers in their patch evaluation standard. For instance, patch writers may focus on whether the submitted patch works, while reviewers also concern about how well the patch works. Therefore, reviewers consider "suboptimal solution" much more decisive for rejecting a patch. As another example, patch writers might pay more attention to the source code than the documentation. However, reviewers turn out to have quite a low tolerance to documentation that is inconsistent with the changed code. As shown in Table V and Figure 6, patch reviewers assign significantly higher decisive scores to "inconsistent or misleading documentation" than patch writers.

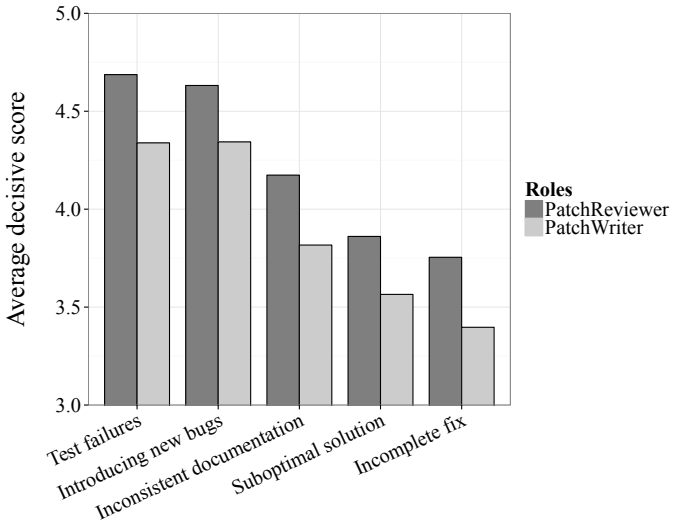

Fig. 6: Patch reviewers and writers assigned significantly different decisive scores to these five patch-rejection reasons. Interestingly, patch reviewers consistently considered these reasons more destructive compared to patch writers.

In general, our result analysis reflects that patch writers and reviewers indeed put different weights on certain patch evaluation criteria, such as *suboptimal solution* and *inconsistent documentation*. In particular, patch reviewers consider them to be much more serious problems than patch writers probably expect.

B. Cost of Patch Review

Our analysis thus far investigates reasons why patches are rejected and how decisive they are. Another important fact is that patch reviewers need to properly justify why they reject a patch in their review comments, for example, why they consider a patch to be suboptimal or incomplete.

However, such a justification can be non-trivial [13]. To understand this non-trivial activity and gain insight into the potential difficulties of patch review, we investigate for each patch-rejection reason, how difficult it is for patch reviewers to justify it. Note that this was included as the second question in our online survey (Section II-B).

We present our results in Figure 7. For each patch-rejection reason, the x-axis shows its average decisive score while the y-axis shows the average difficulty of justifying it. The bubble size represents the percentage of each reason from our manual inspection, which is exactly the same as in Figure 4. We make the following observations with Figure 7.

Although being important, whether a patch introduces new bugs is very difficult to judge: "introducing new bugs" appears in the top-right corner of Figure 7. This means that although it is highly decisive for rejecting patches, it is in fact very difficult to judge. However, a patch that introduces new bugs decreases

the project quality and reliability [25]. Hence, reviewing such a patch is indeed a "high cost, high return" investment: although patch reviewers expend more effort deciding whether the patch introduces new bugs, rejecting it early prevents potentially huge damage on the project's functionality, long-term maintainability or even reputation.

Judging "compilation errors" and "test failures" of patches is highly cost-effective, but these two situations rarely occur: as appeared in the bottom right of Figure 7, "compilation errors" and "test failures" are highly decisive for patch-rejection yet they are relatively easy to judge. However, as shown by their small bubble sizes, we did not observe many occurrences of these two reasons in our manual inspection. One possible explanation is that developers usually check whether their patches compile and pass tests before submission, hence such problems are rarely found by patch reviewers.

Patch review can be particularly beneficial for identifying suboptimal or incomplete patches: "suboptimal solution" is the most frequently observed patch-rejection reason from our manual inspection, as indicated by its largest bubble size. This result is consistent with the study of Bacchelli and Bird, who found that one of the primary incentives of modern code review is to identify alternative solutions [13]. "Incomplete fix" is also a frequently observed reason for patch rejection. We may reasonably speculate that patch review effectively guards against suboptimal or incomplete fix, which could affect the health and maintainability of a project in the long run. However, justifying these two reasons are also difficult (Figure 7), possibly due to the lack of tool support [13].

In general, we have observed a positive correlation between the importance of patch-rejection reasons and the difficulty of judging them. Less decisive reasons are usually easier to judge, such as *missing documentation*, *violating coding styles* and *bad naming*. Highly decisive reasons, on the other hand, are relatively difficult to judge, such as *introducing new bugs*. We only observe a few exceptions, such as *compilation errors* and *test failures*, which are very decisive but also easy to judge as they might have already been taken care of by patch writers.

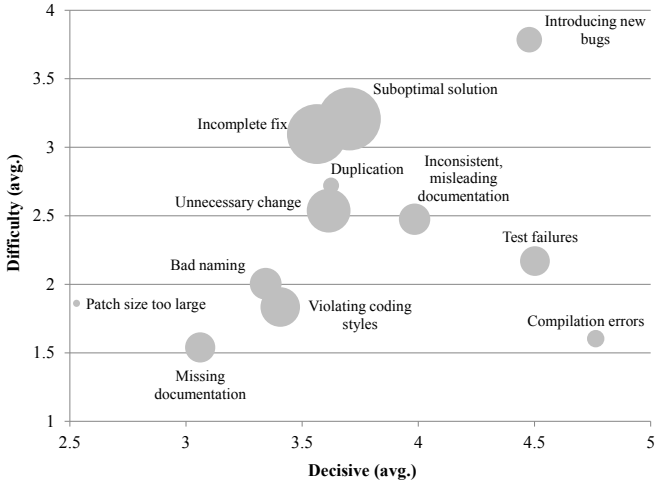

Fig. 7: The x-axis shows the average decisive score for each patch-rejection reason, while the y-axis shows the difficulty of justifying it. The size of the bubble represents the percentage of the corresponding reason identified from our manual inspection, which is the same as in Figure 4.

V. Writing Acceptable Patches (RQ3)

In the prior sections, we discuss several aspects of patch review, such as the common reasons why reviewers reject a patch, the level of difficulty of their judgement, and the potential disconnection between patch reviewers and writers on evaluating patches. Based on these findings on patch rejection, we now propose actionable guidelines for writing acceptable patches.

- Before writing a patch, developers should first make sure that it indeed addresses an issue within the project's scope and relevant to the project's focus (Section III-A).
- Instead of solely focusing on whether the patch fixes the target bug, developers should also be careful of not introducing new bugs, which will be expensive in terms of patch review efforts and the long-term maintainability of the project (Section IV-B).
- Instead of solely focusing on whether the patch works, developers should also consider whether it works well and whether there are better alternative solutions. Patch reviewers appear to be more serious about suboptimal patches than patch writers expect (Section IV-A).
- Developers should check whether their patches are "more" or "less": "more" as in including unnecessary or irrelevant changes and "less" as in not including certain use cases and being incomplete (Section III-A).
- Developers should include or update necessary documentation in their patches. For reviewers, a patch with inconsistent or misleading documentation is highly unacceptable (Section III-B).
- Developers should minimize the high-level impact of a patch on the project schedule and avoid delaying the development of other features (Section III-A).
- Developers should actively communicate with other team members before and after submitting their patches, which increases the chance of the patches being accepted (Section III-A).

VI. Threats to Validity

We manually inspected rejected patches from two open source projects, Eclipse and Mozilla, which use Bugzilla to track issues, patches, and patch reviews. Although both are large projects with a mature issue tracking system, our findings might not generalize to other software projects and those who use different issue tracking or patch review systems. For example, the Gerrit code review system provides the Deckard Autoverifier to automatically verify if a commit builds and merges successfully [26][27]. Only those verified commits proceed to be reviewed by human developers. Hence, for projects that use Gerrit for patch review, we might barely observe patches that are rejected by reviewers for compilation errors. To minimize this threat caused by different projects and patch review mechanisms, we derived the list of patch-rejection reasons from multiple sources. Specifically, the initial

list of patch-rejection reasons identified from our manual inspection was confirmed and supplemented by our online survey respondents and the past literature.

We extracted only the patches with "review-" flags and review comments for our manual inspection, which may not be inclusive of all the rejected patches. Alternative approaches have been proposed to identify rejected patches. For example, Weißgerber et al. considered patches to be accepted if they existed in the repository and rejected otherwise [5]. Also, for projects like Linux, patches without any review comments are implicitly rejected [4]. However, our primary focus was to accurately derive reasons for patch rejection. Hence, we used the explicitly rejected patches that have received review comments for this purpose.

We manually inspected rejected patches and derived patch-rejection reasons. However, this manual process could be subjective. To minimize this threat, two of the authors individually inspected these rejected patches. The inspection results were then compared and discussed with an external CS postgraduate student (Section II-A). Also, as mentioned above, our manual inspection results were further confirmed and complemented by the online developer survey and the past literature.

Our literature survey on patch review might not be comprehensive. We mitigate this threat by adopting the discovering, expanding, and filtering steps as introduced in Section II-C. The first two steps allowed us to cover papers reasonably relevant to patch review and finally reach a convergence, while the last step allowed us to precisely identify papers that discussed reasons for patch rejection.

VII. Related Work

In Section II-C, we introduced studies that explicitly discussed reasons for patch rejection. In this section, we further report studies that have addressed the general quality issue of software patches and artifacts. Fry et al. conducted a human study, in which participants performed tasks to demonstrate their understanding of the functionality and maintainability aspects of software patches [28]. Bettenburg et al. empirically investigated the quality of bug reports from the perspectives of both developers and end-users [8]. Hooimeijer and Weimer proposed a descriptive model of bug report quality, which was used to filter uninformative incoming bug reports or suggest missing features from bug reports [29].

Code and change review are also closely related to our study, as we derived patch-rejection reasons directly from patch review comments. In its early years, code review or inspection was a formal process to detect software defects [30][31][32][33][34]. Studies have observed a 75:25 ratio of maintainability and functional problems fixed in code review [16][17]. Our work complements these studies with more detailed patch-rejection reasons identified from review comments. Bacchelli and Bird proposed that modern code review has become lightweight and focused more on small and incremental code changes [13]. They also found that in addition to finding defects, modern code review provides additional benefits such as knowledge transfer and team transparency [13]. Rigby and Bird studied contemporary peer review by investigating peer review processes in Android, Chromium, Bing, Microsoft Office, MS SQL, and internal projects of AMD [14]. They found that the characteristics of their review process ultimately converge regardless of the differences between these projects.

The human aspects of code review have also been actively studied. Votta argued the number of code review participants in off-line meetings should be minimized to reduce development costs [35]. Asundi and Jayant found that the availability of documentation and coding styles influences non-core members' participation in the patch review process [11]. Baysal et al. suggested that patches written by casual developers should receive extra attention to ensure the quality [2]. They also observed that personal dimensions such as review load and activity have a significant impact on code review outcomes [20]. Rigby and Storey investigated the mechanisms developers use to effectively manage large quantities of code changes and reviews [12]. As a complement to their findings, we also observe that the unresponsiveness of patch writers and a lack of communication can lead to a negative patch review.

Advanced tools have also been proposed to assist code review process. Web-based online code review systems such as Gerrit [26] have been widely adopted. Google introduces a web-based code review system Mondrian [36]. Miller et al. developed a crowd-sourcing-based code review tool, Caesar [37]. This tool slices the entire source code into several code chunks and assigns these chunks to a diverse crowd of reviewers such as alumni and students. Balachandran proposed the Review Bot tool that integrates static analysis output to automatically generate review comments [38].

VIII. Conclusions

In this paper, we empirically investigated reasons why software patches are rejected. Our data sources included 300 rejected Eclipse and Mozilla patches and their review comments, quantitative and qualitative feedback from 246 developers, and a survey of patch-review related studies over the past decade. We derived a comprehensive list of reasons for patch rejection and discovered the discrepancy between patch writers and reviewers in terms of their patch evaluation criteria. We expect our results to be beneficial for both patch writers and reviewers: patch writers can be better aware of reviewers' top "wanted list" and thus better scrutinize their patches to avoid such mistakes; patch reviewers in turn can benefit from the increased quality of patches and reduced review efforts.

Our results also open potential research opportunities. For instance, tool support might be proposed to help reviewers decide whether a patch introduces new bugs, which is currently very difficult to judge. We also plan to exploit our empirical results to build recommendation systems that predict patch review outcomes and suggest potential improvements to patches with human-readable explanations.

Acknowledgment

This work is supported by the Research Grants Council (General Research Fund 613911) of Hong Kong.

REFERENCES

[1] J. Anvik, L. Hiew, and G. C. Murphy, "Coping with an open bug repository," in *Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology eXchange*, ser. eclipse '05. New York, NY, USA: ACM, 2005, pp. 35–39. [Online]. Available: http://doi.acm.org/10.1145/1117696.1117704

[2] O. Baysal, O. Kononenko, R. Holmes, and M. Godfrey, "The secret life of patches: A firefox case study," in *Working Conference on Reverse Engineering*, ser. WCRE '12, 2012, pp. 447 –455.

[3] J. Aranda and G. Venolia, "The secret life of bugs: Going past the errors and omissions in software repositories," in *Proceedings of the 31st International Conference on Software Engineering*, ser. ICSE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 298–308. [Online]. Available: http://dx.doi.org/10.1109/ICSE.2009.5070530

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

[5] P. Weißgerber, D. Neu, and S. Diehl, "Small patches get in!" in *Proceedings of the 2008 international working conference on Mining software repositories*, ser. MSR '08, 2008, pp. 67–76.

[6] B. R. Wiki, https://wiki.mozilla.org/Bugzilla:Review.

[7] A. J. Viera, J. M. Garrett *et al.*, "Understanding interobserver agreement: the kappa statistic," *Fam Med*, vol. 37, no. 5, pp. 360–363, 2005.

[8] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?" in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, ser. SIGSOFT '08/FSE-16, 2008, pp. 308–318.

[9] T. D. LaToza, G. Venolia, and R. DeLine, "Maintaining mental models: a study of developer work habits," in *Proceedings of the 28th international conference on Software engineering*, ser. ICSE '06, 2006, pp. 492–501.

[10] 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*, ser. ICSE '08, 2008, pp. 541–550.

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

[12] 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*, ser. ICSE '11, 2011, pp. 541–550.

[13] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *Proceedings of the 35th International Conference on Software Engineering*, ser. ICSE '13, 2013.

[14] 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.

[15] R. Kula, A. Cruz, N. Yoshida, K. Hamasaki, K. Fujiwara, X. Yang, and H. Iida, "Using profiling metrics to categorise peer review types in the android project," in *Software Reliability Engineering Workshops (ISSREW), 2012 IEEE 23rd International Symposium on*, Nov 2012, pp. 146–151.

[16] M. Mantyla and C. Lassenius, "What types of defects are really discovered in code reviews?" *Software Engineering, IEEE Transactions on*, vol. 35, no. 3, pp. 430–448, May 2009.

[17] A. Z. Moritz Beller, Alberto Bacchelli and E. Jrgens, "Modern code reviews in open-source projects: Which problems do they find fix?" in *Mining Software Repositories (MSR), 2014 11th IEEE Working Conference on*, june 2014, pp. 74 –77.

[18] G. Jeong, S. Kim, T. Zimmermann, and K. Yi, "Improving code review by predicting reviewers and acceptance of patches," 2009.

[19] Y. Jiang, B. Adams, and D. German, "Will my patch make it? and how fast? case study on the linux kernel," in *Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on*, May 2013, pp. 101–110.

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

[21] 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 and software evolution workshops*, ser. IWPSE-Evol '09, 2009, pp. 9–18.

[22] G. G. M. M. Peter C Rigby, Alberto Bacchelli, "A mixed methods approach to mining code review data: Examples and a replication study of multi-commit reviews," 2014.

[23] H. B. Mann, "On a test of whether one of two random variables is stochastically larger than the other," *The Annals of Mathematical Statistics*, vol. 18, no. 1, pp. 50–60, Mar. 1947.

[24] P. Phannachitta, P. Jirapiwong, A. Ihara, M. Ohira, and K.-i. Matsumoto, "An analysis of gradual patch application: A better explanation of patch acceptance," in *Software Measurement, 2011 Joint Conference of the 21st Int'l Workshop on and 6th Int'l Conference on Software Process and Product Measurement (IWSM-MENSURA)*, Nov 2011, pp. 106–115.

[25] Z. Gu, E. T. Barr, D. J. Hamilton, and Z. Su, "Has the bug really been fixed?" in *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ser. ICSE '10. New York, NY, USA: ACM, 2010, pp. 55–64. [Online]. Available: http://doi.acm.org/10.1145/1806799.1806812

[26] Gerrit, http://code.google.com/p/gerrit/.

[27] M. Mukadam, C. Bird, and P. C. Rigby, "Gerrit software code review data from android," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, ser. MSR '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 45–48. [Online]. Available: http://dl.acm.org/citation.cfm?id=2487085.2487095

[28] Z. P. Fry, B. Landau, and W. Weimer, "A human study of patch maintainability," in *Proceedings of the 2012 International Symposium on Software Testing and Analysis*, ser. ISSTA 2012. New York, NY, USA: ACM, 2012, pp. 177–187. [Online]. Available: http://doi.acm.org/10.1145/2338965.2336775

[29] P. Hooimeijer and W. Weimer, "Modeling bug report quality," in *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, ser. ASE '07. New York, NY, USA: ACM, 2007, pp. 34–43. [Online]. Available: http://doi.acm.org/10.1145/1321631.1321639

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

[31] C. Sauer, D. Jeffery, L. Land, and P. Yetton, "The effectiveness of software development technical reviews: a behaviorally motivated program of research," *IEEE Trans. Softw. Eng.*, vol. 26, no. 1, pp. 1 –14, 2000.

[32] H. Siy and L. Votta, "Does the modern code inspection have value?" in *Proceedings. IEEE International Conference on Software Maintenance*, ser. ICSM '01, 2001, pp. 281 –289.

[33] T. Gilb, D. Graham, and S. Finzi, *Software inspection*. Addison-Wesley, 1993, vol. 253.

[34] A. Ackerman, L. Buchwald, and F. Lewski, "Software inspections: an effective verification process," *IEEE Software*, vol. 6, no. 3, pp. 31 –36, 1989.

[35] L. G. Votta, Jr., "Does every inspection need a meeting?" in *Proceedings of the 1st ACM SIGSOFT symposium on Foundations of software engineering*, ser. SIGSOFT '93, 1993, pp. 107–114.

[36] Niall Kennedy. (2006, Dec) Google Mondrian: web-based code review and storage, http://www.niallkennedy.com/blog/2006/11/google-mondrian.html.

[37] Karen A. Frenkel. (Jan, 2013) 'Caesar' Conquers Code Review With Crowdsourcing, http://cacm.acm.org/news/159596-caesar-conquers-code-review-with-crowdsourcing/fulltext.

[38] 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. Piscataway, NJ, USA: IEEE Press, 2013, pp. 931–940. [Online]. Available: http://dl.acm.org/citation.cfm?id=2486788.2486915