The Influence of Non-technical Factors on Code Review

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

***Abstract*—When submitting a patch, the primary concerns of individual developers are "How can I maximize the chances of my patch being approved, and minimize the time it takes for this to happen?" In principle, code review is a transparent process that aims to assess qualities of the patch by their technical merits and in a timely manner; however, in practice the execution of this process can be affected by a variety of factors, some of which are external to the technical content of the patch itself. In this paper, we describe an empirical study of the code review process for WebKit, a large, open source project; we replicate the impact of previously studied factors — such as patch size, priority, and component and extend these studies by investigating organizational (the company) and personal dimensions (reviewer load and activity, patch writer experience) on code review response time and outcome. Our approach uses a reverse engineered model of the patch submission process and extracts key information from the issue tracking and code review systems. Our findings suggest that these non-technical factors can significantly impact code review outcomes.**

***Index Terms*—Code review, non-technical factors, personal and organizational aspects, WebKit, open source software**

I. Introduction

Many software development projects employ code review as an essential part of their development process. Code review aims to improve the quality of source code changes made by developers (as patches) before they are committed to the project's version control repository. In principle, code review is a transparent process that aims to evaluate the quality of patches objectively and in a timely manner; however, in practice the execution of this process can be affected by many different factors, both of technical and non-technical origin.

Existing research has found that organizational structure can influence software quality. Nagappan et al. demonstrated that organizational metrics (number of developers working on a component, organizational distance between developers, organizational code ownership, etc.) are better predictors of defect-proneness than traditional metrics such as churn, complexity, coverage, dependencies, and pre-release bug measures [1]. These findings provide support for Conway's law [2], which states that a software system's design will resemble the structure of the organization that develops it.

In this paper we have performed an empirical study to gain insight into the different factors that can influence how long a patch takes to get reviewed and a patch's likelihood of being accepted. These factors include personal and organizational relationships, patch size, component, bug priority, reviewer/submitter experience, and reviewer load.

From a developer's point of view they are primarily interested in getting their patch accepted as quickly as possible. As such, our research questions in this paper are:

RQ1: *What factors can influence how long it takes for a patch to be reviewed?*
Previous studies have found that smaller patches are more likely to receive faster responses [3]–[5]. We replicate these results and extend the analysis to a number of other potential factors.

RQ2: *What factors influence the outcome of the review process?*
Most studies conclude that small patches are more successful in landing to the project's codebase [3], [4]. A recent study showed that developer experience, patch maturity and prior subsystem churn play a major role in patch acceptance [5]. We further extend these results with additional data that considers various social factors.

In this paper we study the community contributions and industrial collaboration on the WebKit open source project. WebKit is a web browser engine that powers the Apple's Safari and iOS browsers, Google's Chrome and Android browsers, and host of other third-party browsers. WebKit is an interesting project as many of the organizations who collaborate on the project also have competing business interests.

The rest of the paper is organized as follows. We first discuss some background about the WebKit project and its code review process in Section II; this is followed in Section III by a description of the methodology we used in the empirical study. Section IV presents the results of this study, Section V provides interpretation of the results and addresses threats to validity, and Section VI discusses some related work. Finally, Section VII provides a summary of the results and discusses possible future work.

II. Background

WebKit is an HTML layout engine that renders web pages and executes embedded JavaScript code. The WebKit project was started in 2001 as a fork of KHTML. Currently, developers from more than 30 companies actively contribute to this project; Google and Apple are the two primary contributors, submitting 50% and 20% of patches respectively. Individuals from Adobe, BlackBerry, Digia, Igalia, Intel, Motorolla, Nokia, Samsung, and other companies also contribute patches to the project.

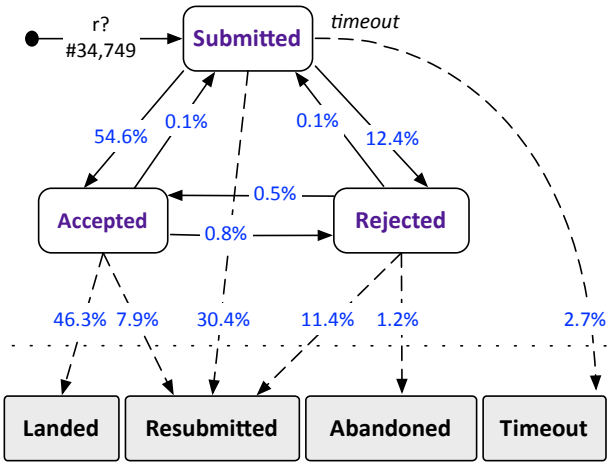

Fig. 1. WebKit's patch lifecycle.

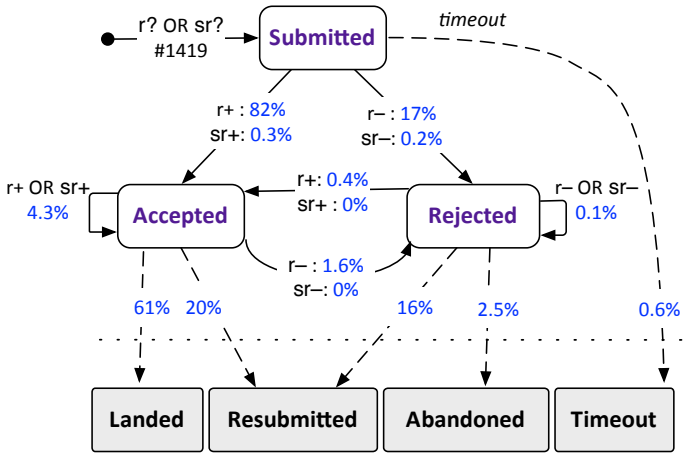

Fig. 2. Mozilla Firefox's patch lifecycle.

The WebKit project employs an explicit code review process for evaluating submitted patches; in particular, a WebKit reviewer must approve a patch before it can land in the project's version control repository. The list of official WebKit reviewers is maintained through a system of voting to ensure that only highly-experienced candidates are eligible to review patches. A reviewer will either accept a patch by marking it `review+` or ask for further revisions from the patch owner by annotating the patch with `review–`. The review process for a particular submission may include multiple iterations between the reviewer and the patch writer before the patch is accepted (lands) in the version control repository.

Since WebKit is an industrial project, we were interested to see whether its code review process is similar to the process adopted by other open source projects. To do so, we extracted the WebKit's patch lifecycle (shown in Figure 1) and compared it with the previously studied patch lifecycle of Mozilla Firefox [6] (shown in Figure 2). The lifecycle demonstrates some interesting transitions that patches go through over their lifecycle that might not otherwise be obvious. For instance, a large proportion of accepted patches are still resubmitted by authors for further revision. We can also see that rejected patches are usually resubmitted, easing concerns that rejecting a borderline patch could cause it to be abandoned.

While patch lifecycle states between WebKit and Firefox remain the same, WebKit has fewer state transitions because the WebKit project does not employ a 'super review' policy. Also, the self edges on the accepted and rejected states are absent in WebKit because while Mozilla patches are often reviewed by two people, WebKit patches receive only individual reviews. Finally, a new edge is introduced between `Submitted` and `Resubmitted`; WebKit developers frequently 'obsolete' their own patches and submit updates before they receive any reviews at all. Comparing of the two patch lifecycles suggests that the WebKit and Firefox code review processes are similar in practice.

In this paper we considered the top five organizations contributing patches to the WebKit repository. Figure 3 provides an overview of the participation of these organizations on the project with respect to the percentage of the total patches they submit and the percentage of the patches they review. It is clear that two companies play a more active role than others; Google dominates in terms of patches written (60% of total project's patches) and patches reviewed (performing 57% of all reviews) while Apple submits 20% of the patches and performs 36% of the reviews.

III. Methodology

In order to investigate our research questions we first extracted the WebKit code review data from Bugzilla, preprocessed it, identified the factors that may affect review delays and outcomes, and performed our analysis.

A. Data Extraction

Every patch contributed to the WebKit project is submitted as an attachment to the project's issue repository[1]; we extracted this data by scraping Bugzilla for all patches submitted between April 12, 2011 and December 12, 2012. We defined the same time interval as the one determined in our previous study on Firefox [6] to be able to compare code review processes of two projects. The data we retrieved consists of 17,459 bugs, 34,749 patches, 763 email addresses, and 58,400 review-related flags. We tracked a variety of information about issues such as name of the person who reported the issue, the date the issue was submitted, its priority and severity, as well as a list of patches submitted for the issue. For each patch we saved information regarding its owner, submission date, whether a patch is obsolete or not, all review-related flags along with the files affected by the patch. For each patch we also recorded the number of lines added and removed along with the number of chunks for each changed file.

All fields in the database, except those related to affected files, were extracted directly from the issue tracker. To create the list of affected files we needed to download and parse the individual patches. Each patch file contains one or more

[1]https://bugs.WebKit.org/

`diff` statements representing changed files. In our analysis we ignored `diff` statements for binary content, e.g., images, and focused on textual `diffs` only. From each statement we extracted the name of changed file, number of lines marked added and removed, and number of code chunks. Here a code chunk is a block of code that represents a local modification to a file as it defined by the `diff` statement. We recorded total number of lines added and lines removed per file in total and not separately for each code chunk. We did not try to derive the number of changed lines from the information about added/removed ones.

Almost every patch in our database affects a file called `ChangeLog` (our analysis found 91 different `ChangeLog` files in the data set). Each `ChangeLog` file contains description of changes performed by the developer for a patch and is prepared by the patch submitter. Although patch files contain `diff` statements for `ChangeLog` files and we parsed them, we eliminated this information when we computed the size of the patch later in our study.

There are three possible flags that can be applied to patches related to code review: `review?` for a review request, `review+` for a review accept, and `review-` for a review reject. For each flag change we also extracted date and time it was made as well as an email address of the person who added the flag.

As the issue tracker uses email addresses to identify people, our initial data set contained many individuals without names or affiliations. Luckily, the WebKit team maintains a file called `contributors.json`[2] that maps various developer email addresses to individual people. We parsed this file and updated our data, reducing the number of people in our database to 747.

We next determined developers' organizational affiliations. First we parsed the "WebKit Team" wiki webpage[3] and updated organizational information in our data. We then inferred missing developers' affiliations from the domain name of their email addresses, e.g., those who have an email at "apple.com" were considered individuals affiliated with Apple. In cases where there was no information about organization available, we performed a manual search on the web. For those individuals where we could not determine an affiliated company, we set `company` field to 'unknown'; this accounted for 18% of all developers and 6% of the patches in our data set.

B. Data Pre-Processing

In our analysis we wanted to focus as much as possible on the key code review issues within the WebKit project. To that end we performed three pre-processing steps on the raw data:

1) We focused only on the patches that change files within the WebCore part of the version control repository. Since WebKit is cross-platform software, it contains a large amount of platform-specific source code. The main parts of WebKit that are not platform-specific are in WebCore; these include features to parse and render HTML and CSS, manipulate the DOM, and parse JavaScript. While this code is actively developed, it is often only developed and reviewed by a single organization (e.g., the Chromium code is only modified by Google developers while the RIM code is only modified by the Blackberry developers). Therefore we looked only at the patches that change non-platform-specific files within WebCore; this reduced the total number of patches considered from 34,749 to 17,170. We also eliminated those patches that had not been reviewed, i.e., patches that had only `review?` flag. This filter further narrowed the input to 11,066 patches.
2) To account for patches that were 'forgotten', we removed slowest 5% of WebCore reviews. Some patches in WebCore are clear outliers in terms of review time; for example, the slowest review took 333 days whereas the median review was only 1.27 hour (≈76.4 minutes). This filter excluded any patch that took more than 120 hours (≈5 days) and removed 553 patches. 10,513 patches remained after this filter was applied.
3) To account for inactive reviewers we removed the least productive reviewers. Some reviewers performed a small number of reviews of WebCore patches. This might be because the reviewer focused on reviewing non-WebCore patches or became a reviewer quite recently. Ordering the reviewers by the number of reviews they performed we excluded those developers performed only 5% of the total reviews. This resulted in 103 reviewers being excluded; the 51 reviewers that remained each reviewed 31 patches or more.

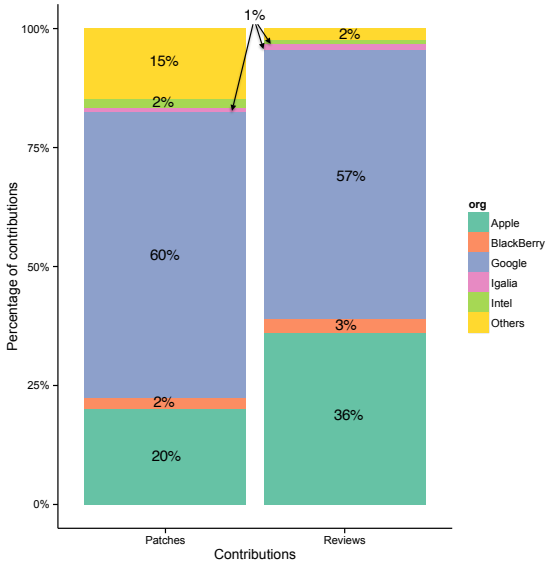

Fig. 3. Overview of the participation of top five organizations in WebKit.

[2]http://trac.WebKit.org/browser/trunk/Tools/Scripts/WebKitpy/common/config/contributors.json

[3]http://trac.WebKit.org/wiki/WebKit%20Team

TABLE I
OVERVIEW OF THE FACTORS AND DIMENSIONS USED.

Independent Factor	Dimension	Description
Patch Size	patch	number of LOC added and removed
Component	patch	top-level module in /WebKit/Source/WebCore/
Priority	bug	assigned urgency of resolving a bug
Organization	organizational	organization submitting or reviewing a patch
Review Queue	personal	number of pending review requests
Reviewer Activity	personal	number of completed reviews
Patch Writer Experience	personal	number of submitted patches

The final dataset[4] consists of 10,012 patches was obtained by taking the intersection of the three sets of patches described above.

C. Determining Independent Factors

Previous research has suggested a number of factors that can influence review response time and outcome [3]–[5]. Table I describes the factors (independent variables) that were considered in our study and tested to see if they have an impact on the dependent variables (*time*, *outcome*, and *positivity*). We grouped the factors into four dimensions: *patch*, *bug*, *organizational*, and *personal*.

In order to define the component a patch modifies, we initially planned to consider the classification of components from the WebKit's bug repository, where each bug is assigned to a specific component. After discussing the WebKit source tree structure with some WebKit developers, we decided to focus only on WebCore patches (as described in the filter above) and identified the modified components from the patches directly rather than use the issue tracker component flag which were often incorrect.

WebKit does not employ any formal patch assignment process. In order to determine review queues of individual reviewers at any given time, we had to reverse engineer patch assignment and answer the following questions:

- *When did the review process start?* We determined the date when a request for a review was made (i.e., `review?` flag was added to the patch). This date was referred as "review start date". While there might be some delay in the actual time a reviewer started working on the patch, we have no practical means of tracking this.
- *Who performed code review of a patch?* The reviewer of a patch is defined as the person who marked the patch with either `review+` or `review-`. Having this we added the assignee to each review request.

We computed a reviewers queue by considering the reviews they eventually completed and walking backwards considering the date the patch was submitted for review and counting the queue length as the number of patches that were 'in flight' for that developer.

D. Data Analysis

Our empirical analysis used a statistical approach to evaluate the degree of the impact of the independent factors on the dependent variables. First, we tested our data for normality by applying Kolmogorov-Smirnov tests [7]. For all samples, the p-value was lower than 0.05 showing that the data is not normally distributed. We also graphically examined how well our data fits the normal distribution using Q-Q plots. Since the data is not normally distributed, we applied non-parametric statistical tests: Kruskal-Wallis analysis of variance [8] for testing whether the samples come from the same distribution, followed by a post-hoc non-parametric Mann-Whitney U (MWW) test [9] for conducting pairwise comparisons.

All our reported results including Figures and Tables are statistically significant with the level of significance defined as p-value ≤ 0.05.

[4]Extracted data is stored in a database available online https://cs.uwaterloo.ca/~obaysal/webkit_data.sqlite

IV. THE CASE STUDY RESULTS

Ultimately, we investigated the impact of seven different factors on the code review process both in terms of response time and review outcome (positivity); this is summarized in Table II.

TABLE II
STATISTICALLY SIGNIFICANT EFFECT OF FACTORS ON RESPONSE TIME AND POSITIVITY.

Independent Factor	Time	Positivity
Patch Size	✓	NA
Priority	✓	✓
Component	✓	×
Organization	✓	✓
Review Queue	✓	✓
Reviewer Activity	✓	×
Patch Writer Experience	✓	✓

A. Patch Size

The size of the patch under review is perhaps the most natural starting point for any analysis, as it is intuitive that larger patches would be more difficult to review, and hence require more time; indeed, previous studies have found that smaller patches are more likely to be accepted and accepted more quickly [3]. We examined whether the same holds for the WebKit patches based on the sum of lines added and removed as a metric of size taken from the patches.

In order to determine the relationship between patch size and the review time, we performed Spearman correlation — a non-parametric test. The results showed that the review time was weakly correlated to the patch size, r=0.09 for accepted patches and r=0.05 for rejected patches, suggesting that patch

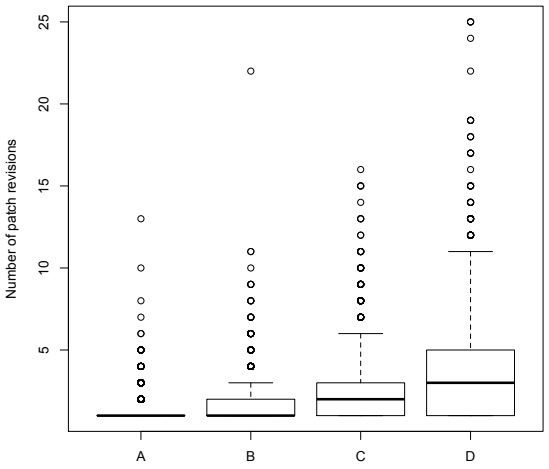

Fig. 4. Number of revisions for each size group.

size and response time are only weakly related, regardless of the review outcome.

With a large dataset, outliers have the potential to skew the mean value of the data set; therefore, we decided to apply two different outlier detection techniques — Pierce's criterion and Chauvenet's criterion. However, we found that removal of the outliers did not improve the results.

Next we split the patches according to their size into four equal groups: A, B, C, and D where each group represents a quarter of the population being sampled. Group A refers to the smallest patches (0–25%) with the average size of 4 lines, group B denoting small-to-medium size changes (25–50%) on average having 17 lines of code, group C consists of the medium-to-large changes (50–75%) with the mean of 54 LOC, and group D represents largest patches (75–100%) with the average size of 432 lines. A Kruskal-Wallis test revealed a significant effect of the patch size group on acceptance time ($\chi^2(3)$=55.3, p-value $<$0.01). Acceptance time for group A (the median time is 39 minutes, the mean is 440 minutes) is statistically different compared to the time for groups B (with the median of 46 minutes and the mean of 531 minutes), C (the median of 48 minutes and the mean of 542 minutes) and D (the median is 64 minutes, the mean time is 545 minutes).

In terms of review outcome, we calculated the positivity values for each group A–D, where we define positivity as `positivity = ∑review+ / (∑review- +∑review+)`. The median values of positivity for groups A–D are 0.84, 0.82, 0.79, 0.74 respectively. Positivity did decrease between the quartiles, matching the intuition that reviewers found more faults with larger patches, although this result was not significant.

However, review time for a single patch is only part of the story; we also wanted to see whether smaller patches undergo fewer rounds of re-submission. That is, we wanted to consider how many times a developer had to resubmit their patch for additional review. We calculated the number of patch revisions for each bug, as well as the size of the largest patch. Figure 4 illustrates the medians of the patch revisions for each size group, the median of the revisions for group A and B is 1, for group C is 2, and for D is 3. The results show that patch size has a statistically significant, strong impact on the rounds of revisions. Smaller patches undergo fewer rounds of revisions, while larger changes have more re-work done before they successfully land into the project's version control repository.

B. Priority

A bug priority is assigned to each issue filed with the WebKit project. This field is created to help developers define the order in which bugs must be fixed[5]. There are 5 different priorities currently in the WebKit ranging from the most important (`P1`) to the least important (`P5`). We were surprised when we computed the distribution of patches among priority levels: `P1` – 2.5%, `P2` – 96.3%, `P3` – 0.9%, `P4` and `P5` – 0.1% each. Looking at these numbers one might speculate that the priority field is not used as intended. Previous work of Herraiz et al. also found that developers use at most three levels of priority and the use of priority/severity fields is inconsistent [10]. The default value for priority is `P2`, which might also explain why the vast majority of patches have this value assigned. Also, in our discussion with WebKit developers we found that some organizations maintain internal trackers that link to the main WebKit bug list; while the WebKit version has the default priority value, the internal tracker maintains the organization's view on the relative priority. In our analysis we discarded priorities `P4` and `P5` because they did not have enough patches.

A Kruskal-Wallis test demonstrated a significant effect of priority on time ($\chi^2(2)$=12.70, p-value $<$0.01). A post-hoc test using Mann-Whitney tests with Bonferroni correction showed the significant differences between P1 and P3 (with median time being 68 and 226 minutes respectively, p-value $<$0.05) and between P2 and P3 (with median time being 62 and 226 minutes respectively, p-value $<$0.01). While patches with priority P2 receive faster response that the ones with P1, the difference is not statistically significant.

To analyze positivity we considered each review by a developer at a given priority and computed their acceptance ratio. To reduce noise (e.g., the data from reviewers who only reviewed one patch at a level and hence had a positivity of 0 or 1), we discarded those reviewers who reviewed 4 patches or fewer for a given priority. We found a statistically significant correlation between priority levels and positivity ($\chi^2(2)$=10.5, p-value $<$0.01). The difference of the review outcome for patches of P1 (median value is being 1.0) compared to the ones of P2 (median is 0.83) is statistically significant (p-value $<$0.01), indicating that patches of higher priority are more likely to land to the project's codebase. Even though reviewers are more positive for patches that are higher priority, we caution about the interpretation of these results because the vast majority of patches are `P2`.

[5]https://bugs.webkit.org/page.cgi?id=fields.html

C. Component

WebCore represents the layout, rendering, and DOM library for HTML, CSS, and SVG. WebCore consists of several primary components:

- `bindings` — houses the language-specific bindings for JavaScript and for Objective-C;
- `bridge` — bridge is about the bridging to the WebKit framework;
- `css` — the CSS back end;
- `dom` — the DOM library;
- `editing` — the editing infrastructure;
- `html` — the HTML DOM;
- `inspector` — web inspector;
- `page` — contains code for the top-level page and frames;
- `platform` — contains platform-specific code, e.g., mac, chromium, android; thus, as mentioned earlier, excluded from our analysis;
- `rendering` — the heart of the rendering engine.

While it is natural assume that some components are more complex than others, we wanted to find out whether contributions to certain components are more successful or reviewed in a timely fashion. To answer this, we selected the components that undergo the most active development: `inspector` (1,813 patches), `rendering` (1,801 patches), `html` (1,654 patches), `dom` (1,401 patches), `page` (1,356 patches), `bindings` (1,277 patches), and `css` (1,088 patches). The difference in the response time between components was statistically significant ($\chi^2(6)=29.9$, p-value <0.01), in particular the `rendering` component takes longer to review (the median time is 101 minutes) compared to `bindings` (72 minutes), `inspector` (58 minutes), and `page` (58 minutes). The difference in reviewing time of patches submitted to the `page` and `dom` components was also significant with the medians being 58 minutes vs. 91 minutes respectively.

Although the positivity values vary among components and range between 0.73–0.84, we found no relation between positivity and the component factor. From the developer's perspective, we can tell that it is more difficult for developers to land a patch to `page` (the value of positivity is 0.73), while patches to `inspector` are more likely to be successful (the value of positivity is 0.84).

D. Review Queue Length

Our previous qualitative study of Mozilla's process management practices found that developers often try to determine current work loads of reviewers prior making a decision about who is the right person to request a review from [11]. Thus, we investigated the relationship between review queue size and review response time expecting to find that reviewers having shorter queues would provide quicker reviews.

We calculated queue sizes for the reviewers at any given time (the process is described in Section III-C). The resulting queues ranged from 0 to 11 patches.

Since the average queue was 0.6 patches, we distributed patches into three groups according to the queue size: shortest queue length ranging from 0–1 patches (group A), medium length consisting of 2–3 patches (group B) and longer queues ranging from 4–11 patches (group C).

We found a significant effect of review queue size on reviewing time ($\chi^2(2)=15.3$, p-value <0.01). The medians of queue size for group A, B and C are being 0, 2, and 5 patches respectively. A post-hoc test showed significant differences between group A and group C (with median time being 63 and 158 minutes respectively, p-value <0.01) and group B and C (with median time being 90 and 158 minutes respectively, p-value <0.05).

Studying the impact of the queue size on the reviewer positivity (with the Kruskal-Wallis effect being $\chi^2(2)=15.8$, p-value <0.01), we found a significant difference between A and C groups (the median positivity being 0.84 and 1 respectively, p-value <0.01), as well as B and C groups (with median positivity being 0.88 and 1.0 respectively, p-value <0.05).

Thus, we found that the length of the review queue influences both the delay in completing the review as well as the eventual outcome: the shorter the queue, the more likely the reviewer is to do a thorough review and respond quickly; and a longer queue is more likely to result in a delay, but the patch has a better chance of getting in.

E. Organization

Many companies that participate in the WebKit development are business competitors. An interesting question is whether patches are considered on their technical merit alone or if business interests play any role in the code review process, for instance by postponing the review of a patch or by rejecting a patch for a presence of minor flaws. While we analyzed all possible pairs of organization (36 of them), for the sake of brevity we discuss only Apple, Google, and 'the rest'.

Figure 5 represents review time for each pair of organizations. The first letter in the label encodes a reviewer's affiliation, the second encodes submitter's affiliation; for example, A-G represents Apple reviewing a Google patch. Analysis of the patches that received a positive review showed that there is a correlation between review time and the organization affiliated with the patch writer.

To identify where the correlation exists, we performed a series of pair-wise comparisons. We discovered that there is a statistically significant difference between how Apple approves their own patches (`A-A`) and how Google approves their own patches (`G-G` column). Another statistically significant difference was found between time Apple takes to accept their own patches and time it takes to accept Google patches (`A-G`). However, we found no statistical difference in the opposite direction — between the time for Google to accept their own patches compared to patches from Apple (`G-A`).

The correlation between review time and company was also found for patches that received a negative review. The pair-wise comparison showed almost the same results: statistical difference between Apple-Apple and Apple-Google, and no statistical difference between Google-Google and Google-Apple. At the same time the difference between Apple-

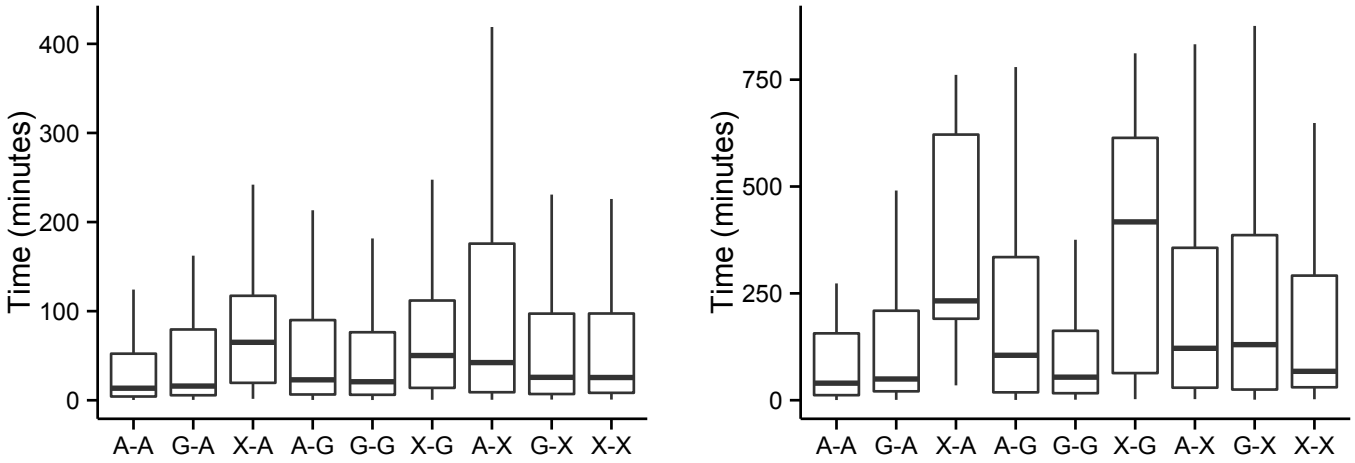

Fig. 5. Acceptance time (left), rejection time (right). Organization: A=Apple, G=Google, X=Rest.

Apple and Google-Google is no longer present. Based on these findings, it appears that Apple treats their own patches differently from external patches, while Google treats external patches more like their own. Pairs involving 'the rest' group exhibited no statistically significant differences for both review decisions.

TABLE III
RESPONSE TIME (IN MINUTES) FOR ORGANIZATIONS.

Reviewer → Writer	Accepted		Rejected	
	Median	Mean	Median	Mean
Apple → Apple	25	392	60	482
Apple → Google	73	617	283	964
Google → Google	45	484	102	737
Google → Apple	42	483	80	543

Since statistical tests can report only a presence of statistical difference, we also report the means and medians of review time required for each company pair (Table III). According to the data, Apple is very fast in reviewing its own patches, but is relatively slow in reviewing Google patches (3–4 times difference in medians, 1.5–2 times difference in means). At the same time Google exhibits the opposite behaviour, i.e., provides faster response to the patches from Apple than their own developers. While both means and medians are almost the same for positive reviews, the median and the mean values of review time for negative review for Apple patches are 20 and 200 minutes less respectively than for Google own patches.

To compute the positivity of various organizations we cleansed the data as we did for the priority analysis above; we removed any reviewer who had reviewed less than 10 patches to avoid an overabundance of positivities of 0 or 1. The box plot with this filter applied is shown in Figure 6. Statistical tests showed that there is a correlation between the outcome of the review and patch owner's affiliation ($\chi^2(2)=10.7$, p-value <0.01). From the pair-wise comparison, we found that there is statistically significant difference between positivity of Apple reviewers towards their own patches (`A-A` column) compared to the patches of both Google (`A-G` column) and 'the rest' (`A-X` column). The other pair that was statistically different is positivity of Google reviewers between their own patches (`G-G` column) and patches from 'the rest' (`G-X` column).

From the quantitative point of view there are some interesting results. First, the positivity of Apple reviewers towards their own patches clearly stands out (the median is ≈0.92). Possible explanations for this include that there is a clear bias among Apple reviewers, or that Apple patches are of extreme quality, or that Apple applies some form of internal code review process. We also observed that both Apple and Google are more positive about their own patches than 'foreign' patches; while this could be a systematic bias, Apple and Google are also the two most experienced committers to WebKit and this may account for this difference. Finally, the positivity of Apple reviewers towards Google patches (the median is ≈0.73) is lower than the positivity of Google reviewers towards Apple patches (the median is ≈0.79).

F. Reviewer Activity

WebCore has 51 individuals performing code reviews of 95% of patches. The breakdown of the reviewers by organization is as follows: 22 Apple reviewers, 19 reviewers from Google, 3 reviewers from BlackBerry, Igalia and Intel are being represented by one reviewer each, and 5 reviewers belong to the group "others". Comparing reviewing efforts, we noticed that while Apple is predominant in the number of reviewers, it only reviews 36% of all patches, while Google developers perform 57% of the total number of reviews. Since WebKit was originally developed and maintained by Apple, it is not surprising that Apple remains a key gatekeeper of what lands to to source code. However, we can see that Google has become a more active contributor on the project, yet has not surpassed the number of Apple reviewers.

In order to find out whether reviewers have an impact on review delay and outcome, for each reviewer we calculated the number of previously reviewed patches and then discretized

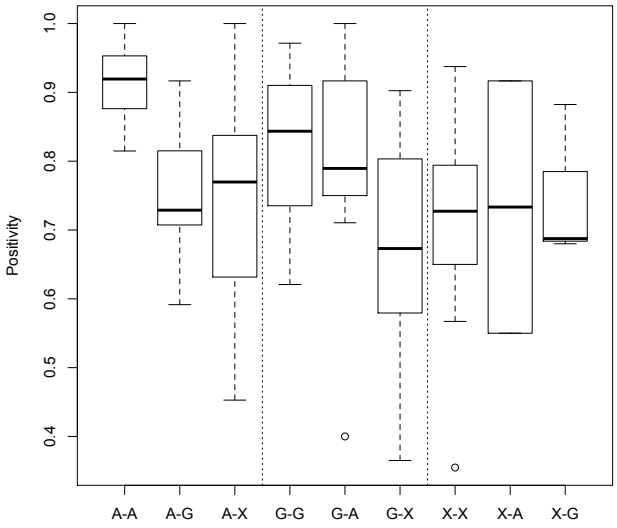

Fig. 6. Positivity values by organization: A=Apple, G=Google, X=Rest.

them according to their reviewing efforts using quartiles. Applying statistical tests we determined that the difference for response time for A and B groups of reviewers (i.e., the less active ones) is statistically significant when compared to C or D groups (i.e., the more active ones). Since the distribution of delays is very skewed, we report both the median and mean values for reviewers' timeliness (see Table IV). The results show that the choice of reviewers plays an important role on reviewing time. More active reviewers provide faster responses (with median being 57 minutes and mean being 496 minutes) compared to the individuals who performed fewer code review (the median for time is 84 minutes and 621 minutes for the mean).

Considering that reviewers' work loads appear to affect their response rate, WebKit contributors may wish to ask the most active reviewers to assess their patches in order to get a quick response. With respect to the question whether there are reviewers who are inclined to be more positive than negative, we found that there is no correlation between the amount of reviewed patches on the reviewer positivity: 0.83 for group A, 0.84 for group B, 0.75 for group C, and 0.83 for group D. This suggests that WebKit reviewers stay true and unbiased in their role of ensuring the quality of code contributions. This observation is important since reviewers serve as gatekeepers protecting the quality of the project's code base.

TABLE IV
RESPONSE TIME (IN MINUTES) FOR PATCH REVIEWERS AND WRITERS.

Group	Reviewer		Writer	
	Median	Mean	Median	Mean
A	84	621	102	682
B	76	634	76	632
C	46	516	43	491
D	57	496	48	478

G. Patch Writer Experience

The contributions to WebCore during the period studied came from 496 individuals among which 283 developers filing 95% of patches (submitting 5 patches or more). Considering our top five organizations, we identified that WebCore patches were submitted by 50 developers from Apple, 219 individuals from Google, 20 BlackBerry developers, 16 developers from Intel, 10 from Igalia and 181 developers come from other organizations.

Noticing good contributor diversity in the WebCore community, we wondered if patches from certain developers have higher chances of being accepted. In order to assess whether developer experience influence review timeliness and acceptance, we performed a similar procedure (as described in IV-F) of calculating the number of submitted changes for each developer and then discretizing patch owners according to their contributions.

We achieved similar results in the differences of response time for A and B groups of submitters (occasional contributors) is statistically significant compared to more experience developers in C or D groups. From Table IV we conclude that more experienced patch writers receive faster responses (with median in group D being 48 minutes and mean being 478 minutes) compared to those who file fewer patches (the median for time in group A is 102 minutes and 682 minutes for the mean).

Investigating the impact of developer experience on positivity of the outcome, we found correlation between two variables ($\chi^2(3)$=17.93, p-value < 0.01). In particular, statistical difference was found between group A (least active developers) and groups C and D (more active developers) with the median positivity values being 1.0, 0.73 and 0.81 respectively, as well as group B (less active developers) compared to the group D (most active ones) with the median positivity being 0.63 and 0.81 respectively. This findings suggest that the WebKit community has a positive incentive for newcomers to contribute to the project as first-patch writers (i.e., group A with the median number of patches submitted = 1) are likely to get a positive feedback. For the developers of group B (where contributions range between 3–6 patches) it is more challenging to get their patches in, while contributing to the project comes with the improved experience of landing patches and as a result with more positive outcomes.

Our findings show that developer experience plays a major role during code review. This supports findings from our previous work, where we have seen faster response time for core developers compared to the casual contributors on the project [6]. This appears to show that active developers are being rewarded with both faster response and more positive review outcome for their active involvement in the project.

H. Multiple Linear Regression Model

To estimate the relationships among factors, we built a multiple linear regression model. Multiple linear regression (MLR) attempts to model the relationship between two or

more explanatory variables and a response variable by fitting a linear equation to observed data [12].

We tested whether the explanatory variables x_1, x_2,... x_p (our factors) collectively have an effect on the response variable y (being review time or outcome), i.e:

H_0: $\beta_1 = \beta_2 = ... = \beta_p = 0$.

The results of the MLR model with respect to time reported the F statistic of 9.096 (p-value <0.01), indicating that we should reject the null hypothesis that the variables size, priority, queue, etc. collectively have no effect on time. The results also showed that the patch writer's experience (p-value <0.01) and affiliation (p-value <0.01) variables are significant controlling for the other variables in the model. While we rejected the null hypothesis, the R-square statistic (i.e., the measure of the regression model's usefulness in predicting outcomes) is close to 0 (R^2=0.01435, $R^2_{adjusted}$=0.01277), indicating that the studied factors have no explanatory power.

Estimating the effect of the factors on the review outcome, we obtained the F statistic of 33.71 (p-value <0.01), and thus rejected the null hypothesis. The most significant factors controlling the other variables in the model remain patch writer's experience (p-value <0.01) and affiliation (p-value <0.01), along with the reviewer load (p-value <0.05) and number of components affected (p-value <0.01). The R-square statistic (R^2=0.0512, $R^2_{adjusted}$=0.04968) still shows that the model has low overall predictive power and that the variables do not account for the the variation in the review outcome in the WebKit project.

While it is not feasible to account for all possible factors that might affect the outcome and interval of the code review process, our findings suggest that among the factors we studied non-technical (organizational and personal) ones are better predictors compared to the traditional metrics such as patch size or component, and bug priority. These findings confirm previous empirical studies on code review [1], [5].

V. Discussion

A. Other Interpretations

Drawing general conclusions from empirical studies in software engineering carries risk: any software development process depends on a potentially large number of relevant contextual variables, which are often non-obvious to outsiders. While our results show that certain non-technical factors have a statistically significant effect on the review time and outcome of patch submissions, understanding and measuring the practical significance of the results remains challenging. Processes and developer behaviour around their contributions to the WebKit project depend on the organization, its culture, internal structure, settings, internal development cycles, time pressures, etc.

Any of these "hidden" factors could potentially influence patch review delays and outcomes; for example, let us consider time pressures. It is our understanding that Apple prefers strict deadlines for shipping hardware, and the supporting software needs to match the projected delivery dates of the new hardware. This results in Apple developers prioritizing internal development goals over external ones, and thus prioritizing patches that help them meet their short-term objectives.

Organizational and geographical distribution of the developers may also provide insights into review delays. WebKit developers at Apple are co-located within the same building which may account for a better visibility of the patches that their co-workers are working on; conversely, WebKit developers at Google tend to be more geographically distributed, which may result in a poorer awareness of the work of others.

In summary, understanding the reasons behind observable developer behaviour requires an understanding of the contexts, processes, organizational and individual factors that can influence code review and its outcome. Thus, while our results may be statistically valid, care must be taken in interpreting their meaning with respect to actual developer behaviour and intent. We consider that much work remains to be done in studying how best to interpret empirical software engineering research within the context of these "hidden" contextual factors.

B. Threats to Validity

Internal validity concerns with the rigour of the study design. In our study, the threats are related to the data extraction process, the selection of the factors that influence code review, and the validity of the results. While we provided details on the data extraction, data filtering and any heuristics used in the study, we also validated our findings with the WebKit developers and reviewers. We contacted individuals from Google, Apple, BlackBerry, and Intel and received insights into their internal processes (as discussed in V-A).

Our empirical study is the subject to *external validity*; we can not generalize our findings to say that both organizational and personal factors affect code review in all open source projects. While we compared WebKit's code review process with the one of Mozilla Firefox and found that its patch lifecycle is similar to open source projects, the fact that WebKit is being developed by competing organizations makes it an interesting case yet a rather obvious exception. Hence, more studies on similar projects are needed.

Statistical conclusion validity refers to the ability to make an accurate assessment of whether independent and dependent variables are related and about the strength of that relationship. In order to determine whether relationships between variables are statistically significant or not we performed null hypothesis testing. We also applied appropriate statistical tests (analysis of variance, post-hoc testing, and Spearman's correlation).

VI. Related Work

Prior work related to this study can be divided into two areas: first, on code review in open source software development; and second, on the effect of organizational structure on the effectiveness. We now provide main findings for each area.

Rigby and German [13] presented a first study that investigated the code review processes in open source projects. They compared the code review processes of four open source projects: GCC, Linux, Mozilla, and Apache. They discovered a number of review patterns and performed a quantitative

analysis of the review process of the Apache project. Later Rigby et al. [4] analyzed 2,603 patches of the Apache open source system and found that small, independent, complete patches are more likely to be accepted. They found that 44% of submitted patches got accepted compared to 46% in our study. In our study, we differentiate negative and positive reviews and investigate what factors may affect time to acceptance or rejection.

Weissgerber et al. [3] performed data mining on email archives of two open source projects to study patch contributions. They found that the probability of a patch being accepted is about 40% and that smaller patches have higher chance of being accepted than larger ones. They also reported that if patches are accepted, they are normally accepted quickly (61% of patches are accepted within three days). Our findings show that 91% of WebKit patches are accepted within 24 hours (ignoring slowest 5% of patches from the analysis).

We have previously studied the code review process of the Mozilla Firefox project, in particular the differences in the patch lifecycles and time taken for each transition for pre- and pos-rapid development models [6]. When analysing Firefox patch acceptance rate, we did not account for the patch size. In this study we investigated the affect of various factors and dimensions on the review time and outcome.

Jiang et al. [5] studied the relation of patch characteristics with the probability of patch acceptance and the time taken for patches to be integrated into the codebase on the example of the Linux kernel. They found that patch acceptance is affected by the developer experience, patch maturity and priori subsystem churn, while reviewing time is impacted by submission time, the number of affected subsystems, the number of suggested reviewers and developer experience. While their patch characteristics do not line up with the factors we studied, we agreed on the same finding that developer experience correlates with the review time. We also found that for the WebKit project response time is affected by the reviewer activity, organization, component and patch size.

Most of the related studies on code review perform mining on project's commit history and thus are not able to reason about negative feedback and rejection interval. We extracted information from the WebKit's issue tracking and code review systems providing a more comprehensive view of the code review process.

While we are not aware of a published work on the WebKit case study, Bitergia's blog provides a general analysis of the WebKit review process, highlighting trends and summaries of how organizations contribute to the project in terms of both patch submission and reviewing activity [14].

VII. Conclusion

The WebKit community is a complex institution in which a variety of organizations that compete at a business level collaborate at a technical level. While it would be ideal for the contributions of these organizations to be treated equally based on their technical merit alone, our results provide empirical evidence that organizational and personal factors influence review timeliness, as well as the likelihood of a patch being accepted. Some factors that influenced the time required to review a patch, such as the size of the patch itself or the part of the code base being modified, are unsurprising and are likely related to the technical complexity of a given change. Other factors did not seem to fit this mould: for example, we found significant differences in how long a patch took to be reviewed based on the organizations that wrote and reviewed a given patch. We found similar effects influencing the chances of a patch being accepted.

Ultimately, the most influential factors of the code review process on both review time and patch acceptance are the organization a patch writer is affiliated with and their level of participation within the project. The more active role a developer decides to play, the faster and more likely their contributions will make it to the code base.

Acknowledgement

We thank the WebKit developers we talked to for their insights into the source code hierarchy and the review process. We also thank Robert Bowdidge for his feedback and comments on a previous draft.

References

[1] N. Nagappan, B. Murphy, and V. Basili, "The influence of organizational structure on software quality: an empirical case study," in *Proc. of the 30th Int. Conference on Software Engeneering*, 2008, pp. 521–530.

[2] M. Conway, "How do committees invent?" *Datamation*, vol. 14, no. 4, pp. 28–31, 1968.

[3] P. Weissgerber, D. Neu, and S. Diehl, "Small patches get in!" in *Proc. of the 2008 Int. Working Conf. on Mining Soft. Repos.*, 2008, pp. 67–76.

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

[5] Y. Jiang, B. Adams, and D. M. German, "Will my patch make it? and how fast? – case study on the linux kernel," in *Proc. of the 10th IEEE Working Conf. on Mining Software Repositories*, San Francisco, CA, US, May 2013.

[6] O. Baysal, O. Kononenko, R. Holmes, and M. Godfrey, "The secret life of patches: A firefox case study," in *Proc. of the 19th Working Conference on Reverse Engineering*, 2012, pp. 447–455.

[7] J. Massey, Frank J., "The kolmogorov-smirnov test for goodness of fit," *Journal of the American Statistical Association*, vol. 46, no. 253, pp. pp. 68–78, 1951.

[8] W. H. Kruskal and W. A. Wallis, "Use of ranks in one-criterion variance analysis," *Journal of the American Statistical Association*, vol. 47, no. 260, pp. pp. 583–621, 1952.

[9] E. Lehmann and H. D'Abrera, *Nonparametrics: statistical methods based on ranks*. Springer, 2006.

[10] I. Herraiz, D. M. German, J. M. Gonzalez-Barahona, and G. Robles, "Towards a simplification of the bug report form in eclipse," in *Proc. of the 2008 Int. Working Conf. on Mining Soft. Repos.*, 2008, pp. 145–148.

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

[12] J. Cohen, *Applied Multiple Regression - Correlation Analysis for the Behavioral Sciences*, 2003.

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

[14] Bitergia, "Reviewers and companies in the webkit project," http://blog.bitergia.com/2013/03/01/reviewers-and-companies-in-webkit-project/, March 2013.