Empir Software Eng (2017) 22:768–817
DOI 10.1007/s10664-016-9452-6

Review participation in modern code review

An empirical study of the android, Qt, and OpenStack projects

Patanamon Thongtanunam[1] · Shane McIntosh[2] · Ahmed E. Hassan[3] · Hajimu Iida[1]

Published online: 20 October 2016

Abstract Software code review is a well-established software quality practice. Recently, Modern Code Review (MCR) has been widely adopted in both open source and proprietary projects. Our prior work shows that review participation plays an important role in MCR practices, since the amount of review participation shares a relationship with software quality. However, little is known about which factors influence review participation in the MCR process. Hence, in this study, we set out to investigate the characteristics of patches that: (1) do not attract reviewers, (2) are not discussed, and (3) receive slow initial feedback. Through a case study of 196,712 reviews spread across the Android, Qt, and OpenStack open source projects, we find that the amount of review participation in the past is a significant indicator of patches that will suffer from poor review participation. Moreover, we find that the description length of a patch shares a relationship with the likelihood of receiving poor reviewer participation or discussion, while the purpose of introducing new features can increase the likelihood of receiving slow initial feedback. Our findings suggest that the

Communicated by: Jeffrey C. Carver

✉ Patanamon Thongtanunam
patanamon-t@is.naist.jp

Shane McIntosh
shane.mcintosh@mcgill.ca

Ahmed E. Hassan
ahmed@cs.queensu.ca

Hajimu Iida
iida@itc.naist.jp

[1] Software Design and Analysis Lab (SDLAB), Nara Institute of Science and Technology, Nara, Japan

[2] Department of Electrical and Computer Engineering, McGill University, Montréal, Canada

[3] Software Analysis and Intelligence Lab (SAIL), Queen's University, Kingston, Canada

patches with these characteristics should be given more attention in order to increase review participation, which will likely lead to a more responsive review process.

Keywords Code review · Review participation · Developer involvement

1 Introduction

Software code review is a well-established software quality practice of having team members critique changes to a software system. Prior work has shown that code reviews can improve the quality of software products by identifying weakness in changes early in the development cycle (Fagan 1999; Shull et al. 2002). Furthermore, recent studies find that code reviews can improve the quality of changes (Bacchelli and Bird 2013; Tsay et al. 2014), and the quality of system design (Morales et al. 2015).

Rigby and Bird (2013) find that current software inspection practices tend to converge on Modern Code Review (MCR), i.e., a lightweight variant of the software inspection process, that has been widely adopted in open source and proprietary projects. Similar to other collaborative processes, the value that is derived from MCR is dependent on the participation of team members. However, the lightweight MCR process often lacks mechanisms for ensuring a base level of review participation, which the formal software inspection process of the past achieved through formalized inspection meetings and review checklists (Fagan 1999). Hence, MCR reviews may not foster a sufficient amount of participation and discussion between author and reviewers.

Furthermore, prior work has shown that review participation has become a key aspect of MCR practices. Rigby et al. (2014) report that the efficiency and effectiveness of code reviews are most affected by the amount of review participation. Recent work finds that the review participation metrics (e.g., the number of involved developers in a review) are associated with the quality of the code review process (Kononenko et al. 2015). Our prior studies also find that a lack of review participation can have a negative impact on long-term software quality (McIntosh et al. 2015; Thongtanunam et al. 2015a). Despite the importance of review participation, little is known about the factors that influence review participation in the MCR process.

In this paper, we set out to investigate the characteristics of patches that suffer from a lack of review participation. In particular, we focus on the characteristics of patches that: (1) do not attract reviewers, (2) are not discussed, and (3) receive slow initial feedback. We measure patch characteristics using 20 patch and MCR process metrics grouped along five dimensions, i.e., patch properties, review participation history, past involvement of an author, past involvement of reviewers, and review environment dimensions. To investigate the relationship between the patch characteristics and the likelihood that a patch will suffer from a lack of review participation, we use contemporary regression modelling techniques that relax the requirement of a linear relationship between explanatory variables and the response, enabling a more accurate and robust fit of the data (Harrell Jr 2002). Through a case study of 196,712 reviews spread across the Android, Qt, and OpenStack open source projects, we address the following three research questions:

(RQ1) What patch characteristics share a relationship with the likelihood of a patch not being selected by reviewers?

We find that the number of reviewers of prior patches, the number of days since the last modification of the patched files share a strong increasing relationship with the likelihood that a patch will have at least one reviewer. The description length is also a strong indicator of a patch that is likely to not be selected by reviewers.

(RQ2) What patch characteristics share a relationship with the likelihood of a patch not being discussed?
We find that the description length, churn, and the discussion length of prior patches share an increasing relationship with the likelihood that a patch will be discussed. We also find that the past involvement of reviewers shares an increasing relationship with the likelihood. On the other hand, the past involvement of an author shares an inverse relationship with the likelihood.

(RQ3) What patch characteristics share a relationship with the likelihood of a patch receiving slow initial feedback?
We find that the feedback delay of prior patches shares a strong relationship with the likelihood that a patch will receive slow initial feedback. Furthermore, a patch is likely to receive slow initial feedback if its purpose is to introduces new features.

Our results lead us to conclude that the review participation history, the description length, the number of days since the last modification of files, the past involvement of an author, and the past involvement of reviewers share a strong relationship with the likelihood that a patch will suffer from poor review participation. Our results highlight the need for patch submission policies that monitor these factors in order to help development teams improve review participation in MCR processes.

1.1 Paper Organization

The remainder of the paper is organized as follows. Section 2 provides an overview of the MCR process and situates this paper with respect to the related work. Section 3 describes the design of our empirical study, while Section 4 presents the results with respect to our three research questions. Section 5 discusses the broader implication of our results. Section 6 discloses the threats to the validity of our empirical study. Finally, Section 7 draws conclusions.

2 Related Work & Research Questions

Modern Code Review (MCR) is a lightweight variant of the software inspection process (Bacchelli and Bird 2013). Figure 1 provides an overview of the MCR process. Broadly speaking, the review process is composed of four main steps:

1. An author uploads a patch (i.e., a set of proposed changes) to a code review tool.
2. An author invites a set of reviewers to critique the patch.
3. Reviewers examine the patch and provide feedback by posting a message to a general discussion thread or inserting inline comments within the patch itself. The author revises the patch to address the feedback from the reviewers and uploads a new revision.
4. When the patch satisfies the requirements of reviewers, reviewers mark the patch as ready for integration into upstream VCS repositories. The patch author or reviewers may also abandon a patch if it does not meet sufficient quality or requires too much rework.

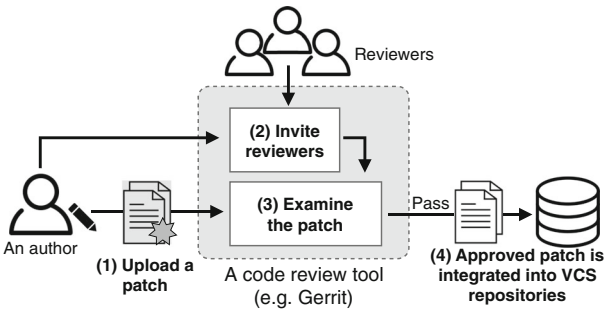


Fig. 1 An overview of the MCR process

2.1 Review Participation

Participation and involvement in software development can have a significant impact on system quality (Abelein and Paech 2013). Code review is a task that requires the involvement of practitioners to critique new software changes (Raymond 1999). Our recent work shows that reviewing expertise which is approximated based on review participation can reverse the association between authoring expertise and defect-proneness (Thongtanunam et al. 2016). Rigby et al. (2014) report that the level of review participation is the most influential factor in the code review efficiency. Several studies have suggested that patches should be reviewed by at least two developers to maximize the number of defects found during the review, while minimizing the reviewing workload on the development team (Sauer et al. 2000; Porter et al. 1998; Rigby and Bird 2013).

Several prior studies find that patches might be ignored during the MCR process (Nurolahzade et al. 2009; Bird et al. 2007). Rigby and storey (2011) indicate that in email-based code reviews, patches are ignored if they do not match with the interests of members of the core development team. In the MCR process, some patches are merged into upstream VCS repositories even though they do not have any participants involved with their reviews apart from the patch author.

Our prior work shows that the number of participants that are involved with a review has a large negative impact on the defect proneness of files in the Qt system, i.e., a file that is examined by more reviewers is less likely to have post-release defects (Thongtanunam et al. 2015a). Bavota and Russo (2015) also find that the patches with low number of reviewers tend to have a higher chance of inducing new bug fixes. Moreover, our prior studies measure review investment (i.e., the proportion of patches that are reviewed and the amount of participation) in a module and examine the impact that review coverage has on software quality (McIntosh et al. 2014, 2015). We find that patches with low review investment are undesirable and have a negative impact on code quality. Hence, in this paper, we try to understand the characteristics of such patches with poor participation. A good understanding of these characteristics will help projects to create mitigation strategies to avoid such poor participation, which in turn would help them to avoid future quality problems (given the empirical link that was established by our prior studies (McIntosh et al. 2014, 2015). Hence, we set out to address the following research question:

RQ1: What patch characteristics share a relationship with the likelihood of a patch not being selected by reviewers?

In addition to the number of participants, the amount of discussion that is associated with a code review also indicates the scrutiny that developers have applied. Careful consideration of the implications of changes improves their overall quality prior to integration (Bacchelli and Bird 2013; Tsay et al. 2014; Thongtanunam et al. 2015a). A review that simply assigns a review score without any suggestion for improvement nor discussion provides little return on code review investment. Recent studies have found that the proportion of changes without review discussion shares a positive relationship with the incidence of both post-release defects (McIntosh et al. 2015), and software design anti-patterns (Morales et al. 2015). However, it is not known whether there are factors that make a review more susceptible to a lackluster discussion. To investigate this, we formulate the following research question:

RQ2: What patch characteristics share a relationship with the likelihood of a patch not being discussed?

While several prior studies extensively investigate the reviewing time (Jiang et al. 2013; Baysal et al. 2015; Gousios et al. 2014), little is known about the factors that share a relationship with *feedback delay*, i.e., the time from a patch submission to initial feedback. A well-functioning code review process should yield responses to a new review request in a timely manner in order to avoid potential problems in the development process (Bettenburg et al. 2013). For example, due to continuous software development practices (Fowler and Foemmel 2006), it is possible that if a patch receives slow initial feedback, it can become outdated, requiring updates to be re-applied (and possibly re-implemented) to the latest version of the system. Moreover, Rigby et al. (2008) suggest that the earlier that a patch is reviewed, the lower the risk of deeply embedded defects. Our recent work also finds that defective files tend to undergo reviews that provide slow initial feedback (Thongtanunam et al. 2015a). To better understand patches that have a long feedback delay, we formulate the following research question:

RQ3: What patch characteristics share a relationship with the likelihood of a patch receiving slow initial feedback?

3 Case Study Design

In this section, we describe the studied projects and present the data preparation, model construction, and model analysis approaches that we use to address our research questions.

3.1 Studied Projects

In order to address our research questions, we perform an empirical study of software projects that actively use MCR for the code review process, i.e., examine and discuss

Table 1 Overview of the studied projects. Android, Qt, and OpenStack satisfy our criteria for analysis, while ITK and VTK do not

Project	Overview		
	Period	Total Patches	Avg. #Patches/Yr
Android	2008/10 - 2014/12 (6 Years)	51,721	8,620
Qt	2011/5 - 2014/12 (4 Years)	99,286	33,095
OpenStack	2011/7 - 2014/12 (4 Years)	136,343	45,447
ITK	2010/8 - 2013/11 (4 Years)	4,305	1,435
VTK	2010/10 - 2013/11 (3 Years)	5,605	1,868

software changes through a code review tool. We began with the review datasets of Android, Qt, and OpenStack projects which are provided by Hamasaki et al. (2013). The review datasets describe patch information, reviewer scoring, the involved personnel, and review discussion history. We also expand the review datasets to include reviews from the VTK and ITK projects. All five projects have been performing code reviews through the Gerrit code review tool for an extended period of time, i.e., more than three years (see Table 1).

Since we focus on projects that actively use MCR, we measure the average number of reviews that have been recorded with the Gerrit code review tool in each year. Table 1 shows that the Android, Qt, and OpenStack projects have a large number of reviews, while the ITK and VTK projects have relatively few reviews. Therefore, we remove the ITK and VTK projects from our study.

The Android Open Source Project[1] is an operating system for mobile devices that is developed by Google. Qt[2] is a cross-platform application and UI framework that is developed by the Digia corporation. OpenStack[3] is an open-source software platform for cloud computing that is developed by many well-known companies, e.g., IBM, VMware, and NEC.

3.2 Data Preparation

To perform our empirical study, we classify patches based on their review participation and extract patch metrics. Figure 2 provides an overview of our data preparation approach. We describe the details of our data preparation approach below.

3.2.1 Selecting Data

To truly understand review participation, we exclude patches that do not satisfy the following criteria:

1. A patch must be submitted during the period when the studied projects actively uses MCR tools.
2. A patch must not be related to VCS bookkeeping activity, such as branch merging.

[1] https://source.android.com/.

[2] http://qt-project.org/.

[3] http://www.openstack.org/.

Fig. 2 An overview of our data preparation approach

For criterion 1, we identify active periods (i.e., years) of MCR usage by computing an active rate, i.e., the number of submitted patches in a year relative to the total number of submitted patches in the whole period that is captured in the datasets. For our analysis, we select the years that have an active rate larger than 10 %. Note that we do not have any gaps in our data, i.e., we find that after the first year that the project has an active rate above 10 %, each following year also has an active rate above 10 %. We focus only on patches that are submitted during the active MCR period because we need to ensure that the low MCR participation are due to patch characteristics and not initial MCR experimentation (like the activity during initial adoption of MCR tools). For criterion 2, we filter out patches that are related to branch merging because such patches are used to perform VCS book-keeping for other patches that have already been revised and integrated. Therefore, such patches generally have little review participation, since the earlier patches have already been reviewed.

3.2.2 Cleaning Data

After selecting patches, we clean the data in order to ensure the accuracy of study results. To do so, we (1) merge the duplicate accounts of a reviewer in the code review systems, and (2) remove auto-generated messages. We describe our approaches below.

Merging the Duplicate Accounts of a Reviewer Similar to Issue Tracking Systems and email discussion threads, Gerrit uses an email address to uniquely identify users. It is possible that a reviewer may have multiple review accounts in the MCR tool due to email aliases of the reviewer. To merge the duplicate accounts, we identify the email aliases using the approaches of Bird et al. (2006). For each reviewer account, we search for the accounts that have a similar name or a similar email name (excluding the email domain) using the generalized Levenshtein edit distance (Ukkonen 1985). We then manually inspect potential duplicates, i.e., those with a Levenshtein edit distance below 0.1.

Removing Auto-Generated Messages Since we will use the messages that are posted in the review discussion thread to measure the participation of reviewers, we need to remove the messages that are left by automated quality gating tools (e.g., static code analyzers) or written by the patch author. We identify the messages that are posted by tools using the accounts of bots in the studied projects. As suggested by Mukadam et al. (2013), we mark the account named "Deckard Autoverifier" as a bot for the Android project. By studying the MCR processes of the Qt and OpenStack projects, we find that the Qt project has Continuous Integration (CI) and Early Warning System (EWS) systems,[4] and the OpenStack project has the Jenkins and Zuul automated testing systems.[5]

[4] https://wiki.qt.io/Qt_Contribution_Guidelines.

[5] http://docs.openstack.org/infra/manual/developers.html#peer-review.

3.2.3 Calculating Patch and MCR Process Metrics

We use 20 patch and MCR process metrics to examine the patches that will suffer from poor review participation. Our metrics are grouped into five dimensions: (1) patch properties, (2) history, (3) past involvement of an author, (4) past involvement of reviewers, and (5) review environment. Table 2 provides the conjecture and the motivating rationale for each of the studied patch and MCR process metrics. Below, we describe the calculation for each of our metrics.

Patch Properties The patch properties dimension measures the change and the information of a patch. To measure the change of a patch, we adopt the change metrics from prior work (Kamei et al. 2013). Churn measures the number of lines added to and removed from modified files. Number of modified files and directories measure the dispersion of a change. Change entropy measures the distribution of modified code across each modified file. Similar to prior work (Hassan 2009), we measure the entropy of a change C as described below:

$$H(C) = -\frac{1}{log_2 n} \sum_{k=1}^{n} (p_k \times log_2 p_k) \quad (1)$$

where n is the number of files included in a patch, p_k is the proportion of change C that impacts file k. The larger the entropy value, the more dispersed that a change is among files.

We also add description length and purpose metrics into this dimension to measure the information that authors provide for the patch. The description length measures how many words an author uses to describe a patch. The purpose indicates the change purpose of a patch. We define the purpose category similar to prior work (Hassan 2008; Mockus and Votta 2000), i.e., documentation, bug fixing, and feature introduction. We classify a patch where its description contains "doc", "copyright", or "license" words as documentation, while a patch where its description contains "fix", "bug", or "defect" words is classified as bug fixing. The remaining patches are classified as feature introduction. A similar approach was used to classify patches in prior studies (Kim et al. 2008; Kamei et al. 2013; McIntosh et al. 2014).

History The history dimension measures the activity of prior patches that modified the same files as the patch under examination. Nagappan et al. (2010) report that the time window of history metrics may have an impact on the prediction models. Therefore, we select a time window based on the development activities of the studied projects. To do so, we study the development cycle by observing the release dates of the studied projects.[6] We find that the studied projects often release a new version every six months. Hence, we measure the history metrics using the six-month period prior to the patch's date of submission. To measure the history metrics for each patch, we focus our analysis on the review activity of patches that (1) occurred on the same branch as the studied patch, and (2) originated

[6] https://en.wikipedia.org/wiki/Android_version_history, https://en.wikipedia.org/wiki/List_of_Qt_releases, https://wiki.openstack.org/wiki/Releases.

Table 2 A taxonomy of patch metrics

Metric	Conjecture	Rationale
Properties Dimension		
Churn	The larger the churn is, the more likely that the patch receives review participation.	Large patches may need more effort to review (Mishra and Sureka 2014; Rigby et al. 2008; Rigby et al. 2014).
Number of Modified Files	The more files that are changed in this patch, the more likely that the patch will suffer from poor review participation.	Patches where their changes scatter across a large number of files or directories may need more effort to review. Finding reviewers who have knowledge for such changes is difficult as well. Therefore, it is more likely that the patch will suffer from poor review participation.
Number of Modified Directories	The more directories that are impacted by this patch, the more likely that the patch will suffer from poor review participation.	
Entropy	The more scattered the changes in this patch are, the more likely that the patch will suffer from poor review participation.	
Description Length	The longer description in the patch, the less likely that the patch will suffer from poor review participation.	Patches with a descriptive subject and a well explained change log message would be able to draw the attention of reviewers (Rigby and Storey 2011).
Purpose	A patch that introduces new functionality is more likely to receive slow initial feedback than a patch for another purposes.	Patches that introduce new features may require more effort to examine than patches for other purposes.
History Dimension		
Number of Days since the Last Modification	A patch containing files that have been recently changed is more likely to receive responsive review participation.	Recently changed files could be the files on which developers are currently working (i.e., more knowledgeable).
Total Number of Authors	The more developers who have written patches made to the modified files, the more likely that the patch receives responsive review participation.	A reviewer is likely to be one of the authors of a frequently changed file.
Number of Prior Defects	A patch containing files that have many defects is more likely to receive responsive review participation.	Files that have historically been defective may require additional attention during the code review process (Thongtanunam et al. 2015a).
Number of Reviewers of Prior Patches	The more reviewers who have examined prior patches, the more likely the patch receive responsive participation.	Files that have been previously examined by many reviewers would have the likelihood that one of those reviewers is a reviewer of this patch.

Table 2 (continued)

Metric	Conjecture	Rationale
Discussion Length of Prior Patches	The longer discussion that the modified files have received in prior patches, the more likely that the patch receive responsive participation.	Files that have received long discussion in prior patches could be complicated. Hence, they may require additional attention during the code review process.
Feedback Delay of Prior Patches	The longer the feedback delay in prior patches is, the more likely that the patch will suffer from poor review participation.	Files that often receive slow initial feedback can be those with less priority than other files. Hence, they may receive little review participation.
Past Involvement of an Author Dimension		
Number of Prior Patches of an Author	The more prior patches that the author has either written or examined, the more likely that the patch receives review participation.	Patches written by inexperienced authors are more likely to receive little review participation, since the authors are not familiar with the project and may not know who should be invited to review the changes (Bosu and Carver 2013).
Recent Patches of an Author	The more the recent patches of an author are, the more likely that the patch receives review participation.	
Number of Directory Patches of an Author	The more the directory patches of an author are, the more likely that the patch receives review participation.	
Past Involvement of Reviewers Dimension		
Number of Prior Patches of Reviewers	The more prior patches that the reviewers have either written or examined, the more likely that the patch receives review participation.	Patches reviewed by experienced reviewers are very likely to receive prompt initial feedback and long discussion, since such reviewers have a good understanding and a strong familiarity of the context in which a change is being made (Rigby et al. 2008; Rigby et al. 2012; Baysal et al. 2015; Thongtanunam et al. 2015b).
Recent Patches of Reviewers	The more the recent patches of reviewers are, the more likely that the patch receives review participation.	
Number of Directory Patches of Reviewers	The more the directory patches of reviewers are, the more likely that the patch receives review participation.	
Review Environment Dimension		
Overall Workload	The more the overall workload of the project is, the more likely that the patch will suffer from poor review participation.	Reviewers can be burdened with a large workload. Therefore, it is more likely that patches will receive little review participation if they are submitted at a time when a project has a large review workload (Rigby and Storey 2011; Baysal et al. 2012).

Table 2 (continued)

Metric	Conjecture	Rationale
Directory Workload	The more the directory workload is, the more likely that the patch will suffer from poor review participation.	

on other branches, but have been merged into the same branch as the studied patch. The number of days since the last modification measures how long it has been since the files that were modified in the patch were last modified. The total number of authors counts how many people have submitted patches that impact the same files as the patch under examination. The number of prior defects counts the number of prior bug-fixing patches that impact the same files as the patch under examination.

Furthermore, we adopt the hypothesis of prior work to estimate past tendencies (Tantithamthavorn et al. 2015). We measure the past tendency of review participation using three metrics, i.e., the number of reviewers, discussion length, and feedback delay of prior patches that have been applied to the same files as the files in the patch under examination. The number of reviewers of prior patches measures the median number of reviewers who have posted messages or a reviewing score in the reviews of prior patches that impact the same files as the patch under examination. The discussion length of prior patches measures the median number of messages that are posted in the reviews of prior patches that impact the same files as the patch under examination. The feedback delay of prior patches measures the median of feedback delays that the reviews of prior patches had received. We use the median value because we find that the distributions of the history data do not follow normal distribution (i.e., the p-values of Shapiro-Wilk tests are lower than 0.05 for all of the studied patches).

Past Involvement of an Author The past involvement of an author dimension measures the activity in which an author has been involved before making the patch under examination. To calculate the past involvement of an author metrics, we use the same approach as the history dimension to collect the activity in which an author has been involved. The number of prior patches of an author counts the number of submitted or reviewed patches by the author prior to the patch under examination. The recent patches of an author is a variant of the number of prior patches of an author, which is weighted by the age of the prior patches. The more recent patches are given a higher weight than the less recent ones. Similar to prior work (Kamei et al. 2013), we measure the recent patches for an author as described below:

$$RC = \sum_{m \in M} \frac{N_c}{m} \tag{2}$$

where N_c is the number of patches that have been submitted or reviewed by the author in the past month m in the time window M (i.e., six-month period). Similar to prior work (Kamei et al. 2013), we also measure a higher level of experience for an author using the number of directory patches of an author metric. The number of directory patches of an author measures how many prior patches modify code in the same directories as the patch under examination, and were submitted or reviewed by the author.

Past Involvement of Reviewers The past involvement of reviewers dimension measures the activity that the patch reviewers have been involved with prior to the patch under examination. Similar to past involvement of an author dimension, we measure the number of prior patches, recent patches, and the number of directory patches of reviewers metrics.

Review Environment The review environment dimension measures the code review activity that occurred during the same period as the patch being submitted. To measure review environment metrics, we use a 7-day period prior to the time that a patch is submitted. We select the 7-days period as our time window because we find that the time from patch submission to review completion is shorter than 3, 4, and 11 days for 75 % of the reviews in the Android, Qt, and OpenStack projects, respectively. The overall workload metric counts how many patches are submitted to the code review tool. The directory workload metric counts how many prior patches modify code in the same directories as the patch under examination.

3.3 Model Construction

We build logistic regression models to determine the likelihood of a patch suffering from poor review participation. Logistic regression models are commonly used to study how explanatory variables are related to a dichotomous response variable. In our study, we use our patch and MCR process metrics as explanatory variables, while the response variable is assigned the value of TRUE if a patch suffered from poor review participation, and FALSE otherwise.

Similar to prior work (McIntosh et al. 2015), we adopt the model construction and analysis approaches of Harrell Jr (2002, p. 79) to allow nonlinear relationships between explanatory and response variables to be modelled. Furthermore, these techniques can enable a more accurate and robust fit of the data, while carefully considering the potential for overfitting (i.e., a model is too specifically fit to the training dataset to be applicable to other datasets). An overfit model will overestimate the performance of the model and exaggerate spurious relationships between explanatory and response variables. Figure 3 provides an overview of the three steps in our model construction approach. We describe below each step in our approach.

3.3.1 Correlation & Redundancy Analysis

Explanatory variables that are highly correlated with each other can interfere with the results of model analysis. Hence, we measure the correlation between explanatory variables

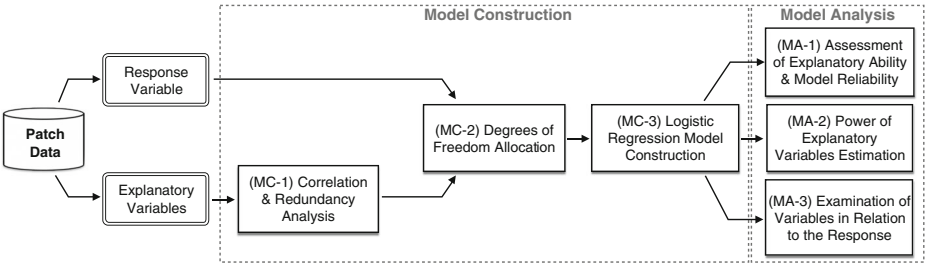


Fig. 3 An overview of our model construction and analysis approaches

using Spearman rank correlation tests (ρ). We then use a variable clustering analysis technique (Sarle 1990) to construct a hierarchical overview of the correlation and remove explanatory variables with a high correlation. According to Hinkle et al (1998, p. 120), Spearman correlation coefficient values that are greater than 0.7 are considered to be strong correlation. Hence, we select $|\rho| = 0.7$ as our threshold for removing highly correlated variables. We perform this analysis iteratively until all clusters of surviving variables have $|\rho|$ values below 0.7.

Some explanatory variables are not highly correlated but can still be redundant, i.e., variables that do not have a unique signal from the other explanatory variables. Redundant variables in an explanatory model will distort the modelled relationship between the explanatory and response variables. To detect redundant variables, we use the `redun` function in the `rms` R package (Harrell Jr 2015) to fit models that explain each explanatory variable using the remaining explanatory variables. We then remove the explanatory variables where models are fit with an R^2 value greater than 0.9 (the default threshold of the `redun` function).

3.3.2 Degrees of Freedom Allocation

Before allocating the degrees of freedom to the model, we have to ensure that our model will not be overfit. A model can be overfit if it uses degrees of freedom (e.g., explanatory variables) more than a dataset can support. To minimize the risk of overfitting, we need to estimate a budget for degrees of freedom, i.e., number of explanatory variables that a dataset can support, before fitting a model. As suggested by Harrell Jr (2002, p.60), we set a budget for degrees of freedom to be $\frac{\min(T,F)}{15}$, where T is the number of rows where the response variable is TRUE, and F is the number of rows where the response variable is FALSE.

In order to allow nonlinear relationships between explanatory and response variables to be modelled, we must decide how to allocate our budgeted degrees of freedom to each of our explanatory variables. To allocate the degrees of freedom most effectively, the explanatory variables that have more potential for sharing nonlinear relationship with the response variable should be allocated more degrees of freedom than the explanatory variables that have less potential.

We measure the potential for nonlinearity in the relationship between explanatory and response variables using a calculation of the Spearman multiple ρ^2. A large Spearman multiple ρ^2 score indicates that there is potential for a strong nonlinear relationship between an explanatory variable and the response variable. Taking the budgeted degrees of freedom into account, we allocate the degrees of freedom to explanatory variables according to their Spearman multiple ρ^2 values, i.e., variables with larger ρ^2 values are allocated more degrees of freedom. Nevertheless, we limit the maximum degrees of freedom that we allocate to any given explanatory variable to five in order to minimize the risk of overfitting (Harrell Jr 2002, p. 23).

3.3.3 Logistic Regression Model Construction

After removing the highly correlated and redundant variables and allocating the degrees of freedom to the surviving explanatory variables, we fit our logistic regression models to the data. We use the restricted cubic splines of the `rcs` function in the `rms` R package (Harrell Jr 2015) to fit the allocated degrees of freedom to the explanatory variables. We use the restricted cubic splines because the smooth nature of cubic curves will more realistically fit

natural phenomena than linear splines, which introduce abrupt changes in direction (Harrell Jr 2002, p. 20).

3.4 Model Analysis

Once the logistic regression model has been constructed, we analyze the model in order to understand the relationship between the explanatory variables (i.e., patch and MCR process metrics) and the response variable (i.e., whether a patch had review participation or not). Figure 3 shows our three steps of model analysis. We describe below each step of our model analysis approach.

3.4.1 Assessment of Explanatory Ability & Model Reliability

To evaluate the performance of our models, we use the Area Under the receiver operating characteristic Curve (AUC) (Hanley and McNeil 1982). AUC measures how well a model can discriminate between the potential responses. It is computed by measuring the area under the curve that plots the true positive rate against the false positive rate while varying the threshold that is used to determine whether a patch is classified as receiving review participation or not. An AUC value of 1 indicates perfect discrimination, i.e., perfect separation of patches that receive review participation and those that do not, while an AUC value of 0.5 indicates that the model does not discriminate better than random guessing.

Although the AUC can measure the explanatory power, it may overestimate the performance of the model if it is an overfit model. To evaluate the reliability of our models, we estimate the optimism of the AUC using a bootstrap-derived approach (Efron 1986). First, the approach trains a model using a bootstrap sample, i.e., a dataset sampled with replacement from the original dataset, which has the same population size as the original dataset. Then, the optimism is estimated using the difference in performance between the bootstrap model when applied to the original dataset and the bootstrap sample. Finally, the approach is repeated 1,000 times in order to compute the average optimism. The smaller the average optimism is, the more reliable the performance estimates of the original fit are.

3.4.2 Power of Explanatory Variables Estimation

Similar to prior work (McIntosh et al. 2015), we use Wald statistics to estimate the impact that each explanatory variable has on the model's performance. Since the explanatory variables that were assigned additional degrees of freedom are represented in the model by multiple terms, Wald statistics are used to jointly test all model terms that relate to a given explanatory variable. For the tests, we use the `anova` function in the `rms` R package (Harrell Jr 2015) to estimate the relative contribution (Wald χ^2) and the statistically significance (p-value) of each explanatory variable in the model. The larger the Wald χ^2 value is, the larger the explanatory power that a particular explanatory variable contributes to the performance of the model.

3.4.3 Examination of Variables in Relation to the Response

The power of explanatory variables indicates the magnitude of the impact that an explanatory variable has on model performance, yet it does not provide a notion of the direction or the shape of the relationship between the explanatory variables and the response. To better understand the direction and shape of these relationships, we plot the odds value produced

by our models against an explanatory variable while holding the other explanatory variables at constant values. We use the `Predict` function in the `rms` R package (Harrell Jr 2015) to compute and plot the odds values for various explanatory variables.

In addition, we estimate the partial effect that explanatory variables have on the response using odds ratio (Harrell Jr 2002, p. 220). Odds ratio indicates the change to the likelihood of a patch that will suffer from poor review participation when the value of an explanatory variable under study increases. The larger the odds ratio is, the larger the partial effect that the explanatory variable has on the likelihood. We analyze the relative percentage that the odds has changed corresponding to the changed value of the explanatory variable, while holding the other explanatory variables at constant values. Using the `summary` function in the `rms` R package (Harrell Jr 2015), the partial effect is estimated based on the odds difference of the inter-quartile range for continuous variables, and the odds difference between each category value and the mode (i.e., the most frequently occurring category) for categorical variables. A positive partial effect indicates an increasing relationship between the explanatory variable and the response, while a negative partial effect indicates an inverse relationship. The magnitude of a partial effect indicates the amount that the odds value in our models will change according to the shifted value of the explanatory variable.

4 Case Study Results

In this section, we present the results of our study with respect to our research questions. Table 1 provides a summary of the patch and review data that we selected according to our data preparation approach. Table 3 shows distributions of data for the patch metrics that we measure in the studied datasets. For each research question, we discuss our: (a) model construction procedure and (b) model analysis results.

(RQ1) What Patch Characteristics Share a Relationship with the Likelihood of a Patch not Being Selected by Reviewers?

In the Gerrit process, some patches can be merged into upstream VCS repositories even though these patches do not have any participant apart from the patch author. For example, in the review ID 29921 in the Qt project, no third-party reviewers participated in the review by neither voting a score or posting a message, although the patch author had invited a reviewer.[7] Furthermore, our prior study has shown that the number of patches that do not have reviewers providing feedback to the reviews shares a relationship with defect-proneness (McIntosh et al. 2014). Hence, to address our RQ1, we identify the patches that do not attract reviewers by counting the number of unique reviewers who participated in a review by either posting a message or assigning a reviewing score. We classify patches into two categories — those that attract at least one reviewer and those that do not.

Table 4 provides the number of patches from our patch classification. Below, we present and discuss the results of our model construction and analysis.

4.1 (RQ1-a) Model Construction

According to our model construction approach (cf. Fig. 3), the response variable is set to TRUE if a patch is not reviewed by another developer and FALSE otherwise. We use our

[7] https://codereview.qt-project.org/#/c/29921/.

Table 3 Descriptive statistics of the studied patch metrics

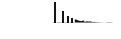
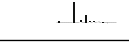
		Android	Qt	OpenStack			Android	Qt	OpenStack
Patch Properties Dimension					History Dimension				
Churn	1st Qu.	5.00	4.00	5.00	#Prior defects	1st Qu.	0.00	0.00	0.00
	Median	20.00	15.00	21.00		Median	1.00	1.00	3.00
	Mean	4434.00	603.00	767.00		Mean	5.69	4.01	10.48
	3rd Qu.	96.00	63.00	90.00		3rd Qu.	4.00	4.00	10.00
	Histogram					Histogram			
#Modified files	1st Qu.	1.00	1.00	1.00	#Days since the last modification	1st Qu.	1.47	1.26	1.18
	Median	2.00	2.00	2.00		Median	11.31	8.19	5.45
	Mean	27.02	10.81	4.81		Mean	55.11	41.83	28.62
	3rd Qu.	4.00	4.00	3.00		3rd Qu.	109.24	49.48	22.52
	Histogram					Histogram			
#Modified directories	1st Qu.	1.00	1.00	1.00	#Total authors	1st Qu.	0.00	0.00	1.00
	Median	1.00	1.00	1.00		Median	1.00	2.00	4.00
	Mean	4.10	3.09	2.57		Mean	2.70	3.31	13.40
	3rd Qu.	2.00	2.00	2.00		3rd Qu.	3.00	4.00	13.00
	Histogram					Histogram			
Entropy	1st Qu.	0.00	0.00	0.00	#Reviewers of prior patches	1st Qu.	0.00	1.00	1.50
	Median	0.22	0.10	0.49		Median	1.00	1.00	3.00
	Mean	0.90	0.83	0.85		Mean	0.92	1.29	2.85
	3rd Qu.	1.46	1.31	1.36		3rd Qu.	1.00	2.00	4.00
	Histogram					Histogram			

Table 3 (continued)

		Android	Qt	OpenStack			Android	Qt	OpenStack
Description length	1st Qu.	11.00	10.00	13.00	Discussion length of prior patches	1st Qu.	0.00	0.00	1.00
	Median	22.00	19.00	28.00		Median	0.00	0.50	2.00
	Mean	35.61	28.74	36.46		Mean	1.23	1.90	4.15
	3rd Qu.	45.00	36.00	47.00		3rd Qu.	1.00	2.00	5.00
	Histogram					Histogram			
Purpose	BUG-FIX	13360.00	24827.00	39405.00	Feedback delay of prior patches	1st Qu.	0.00	0.35	0.73
	Document	757.00	6830.00	8167.00		Median	0.64	1.53	2.32
	Feature	19852.00	34471.00	48043.00		Mean	31.91	24.17	11.22
						3rd Qu.	3.19	7.87	6.45
						Histogram			
Past Involvement of an Author Dimension					Past Involvement of Reviewers Dimension				
#Prior patches of an author	1st Qu.	3.00	13.00	4.00	#Prior patches of reviewers	1st Qu.	0.00	14.00	34.00
	Median	22.00	60.00	24.00		Median	21.00	84.00	171.00
	Mean	83.36	128.50	96.02		Mean	101.80	179.80	331.60
	3rd Qu.	105.00	168.00	114.00		3rd Qu.	139.00	245.00	495.00
	Histogram					Histogram			
Recent patches of an author	1st Qu.	0.33	2.67	1.00	Recent patches of reviewers	1st Qu.	0.00	2.45	13.88
	Median	8.15	23.00	10.17		Median	7.88	33.00	72.05
	Mean	35.27	53.65	40.03		Mean	43.03	73.73	135.41
	3rd Qu.	48.67	74.40	49.75		3rd Qu.	60.50	105.00	200.50

Table 3 (continued)

		Android	Qt	OpenStack			Android	Qt	OpenStack
	Histogram					Histogram			
#Directory patches of an author	1st Qu.	0.00	1.00	2.00	#Directory patches of reviewers	1st Qu.	0.00	1.00	9.00
	Median	8.00	13.00	11.00		Median	4.00	15.00	48.00
	Mean	78.34	116.60	68.44		Mean	82.75	104.90	161.80
	3rd Qu.	52.00	76.00	51.00		3rd Qu.	44.00	88.00	164.00
	Histogram					Histogram			
Review Environment Dimension									
Overall workload	1st Qu.	221.00	511.00	1220.00	Directory workload	1st Qu.	1.00	1.00	2.00
	Median	394.00	580.00	1524.00		Median	3.00	5.00	8.00
	Mean	369.40	587.20	1430.00		Mean	10.85	10.66	63.79
	3rd Qu.	477.00	674.00	1626.00		3rd Qu.	10.00	12.00	33.00
	Histogram					Histogram			

Histograms are in a log scale

Table 4 Patch data for the study of RQ1

Project	Patch data		
	Studied period	#Patches that did not attract reviewers (TRUE class)	#Total Patches
Android	2012/01 - 2014/12	8,356	33,969
Qt	2012/01 - 2014/12	7,234	66,128
OpenStack	2013/01 - 2014/12	8,360	95,615

patch and MCR process metrics that are described in Table 2 as explanatory variables. However, we did not use the past involvement of reviewers metrics, since past involvement of reviewers cannot be measured in the patches that do not have reviewers. We then construct logistic regression models, which we describe in detail below.

(MC-1) Correlation & Redundancy Analysis Before constructing a model, we remove the explanatory variables in Table 2 that are highly correlated with one another based on hierarchical clustering analysis. If a cluster of explanatory variables have a Spearman's $|\rho| > 0.7$, we select one variable from the cluster. For example, Fig. 4 shows the hierarchical clustering of explanatory variables in the Android dataset. There are three clusters of variables that have a Spearman's $|\rho| > 0.7$, i.e., (1) the number of files and the number of directories, (2) the number of prior patches, recent patches, and directory patches of an author, (3) the number of prior defects and the total number of authors, and (4) the number of reviewers and discussion length of prior patches. For the first and second clusters, we select the number of files and the prior patches of an author as the representative variables because they are simpler to calculate than the other variables in their clusters. For the third cluster, we select the number of prior defects and remove the total number of authors since the distribution of the number of prior defects is less skewed than the total number of authors. For the fourth cluster, both explanatory variables (the number of reviewers and discussion length of prior patches) are participation tendency metrics, which are simple to calculate. We select the number of reviewers of prior patches as the representative

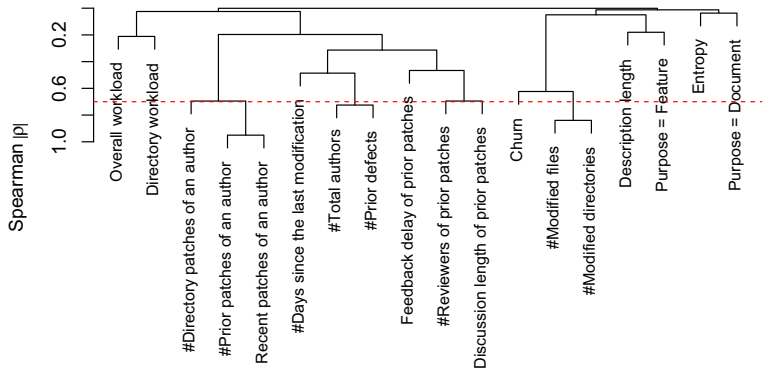


Fig. 4 Hierarchical clustering of variables according to Spearman's $|\rho|$ in the Android dataset (RQ1). The dashed line indicates the high correlation threshold (i.e., Spearman's $|\rho| = 0.7$)

Table 5 Statistics of the logistic regression models for identifying patches that do not attract reviewers (RQ1)

		Android		Qt		OpenStack	
AUC (|Optimism|)		0.72 (0.002)		0.70 (0.001)		0.74 (0.001)	
Budgeted D.F.		557		482		557	
Spent D.F.		14		16		17	
Wald χ^2		2,218***		2,910***		3,804***	
		Overall	Nonlinear	Overall	Nonlinear	Overall	Nonlinear
Patch Properties Dimension							
Churn	D.F.	1	–	1	–	1	–
	χ^2	0 %°		0 %°		0 %***	
#Modified files	D.F.	†		†		†	
	χ^2						
#Modified directories	D.F.	†		†		†	
	χ^2						
Entropy	D.F.	1	–	1	–	1	–
	χ^2	0 %°		1 %***		1 %***	
Description length	D.F.	1	–	2	1	2	1
	χ^2	0 %*		15 %***	13 %***	5 %***	5 %***
Purpose	D.F.	2	–	2	–	2	–
	χ^2	1 %***		1%***		1%***	
History Dimension							
#Days since the last modification	D.F.	1	–	1	–	1	–
	χ^2	16 %***		30 %***		15%***	
#Total authors	D.F.	†		1	–	†	
	χ^2			0 %°			
#Prior defects	D.F.	1	–	1	–	1	–
	χ^2	0 %°		0%°		0 %**	
#Reviewers of prior patches	D.F.	2	1	3	2	4	3
	χ^2	81 %***	26 %***	69 %***	42 %***	72 %***	32 %***
Discussion length of prior patches	D.F.	†		†		†	
	χ^2						
Feedback delay of prior patches	D.F.	2	1	1	–	2	1
	χ^2	1 %***	1 %***	0 %°		1 %***	0 %***
Past Involvement of an Author Dimension							
#Prior patches of an author	D.F.	1	–	1	–	1	–
	χ^2	0 %**		0 %***		0 %**	
Recent patches of an author	D.F.	†		†		†	
	χ^2						
#Directory patches of an author	D.F.	†		†		†	
	χ^2						
Review Environment Dimension							
Overall workload	D.F.	1	–	1	–	1	–

Table 5 (continued)

		Android		Qt		OpenStack	
AUC (|Optimism|)		0.72 (0.002)		0.70 (0.001)		0.74 (0.001)	
Budgeted D.F.		557		482		557	
Spent D.F.		14		16		17	
Wald χ^2		2,218***		2,910***		3,804***	
		Overall	Nonlinear	Overall	Nonlinear	Overall	Nonlinear
Directory workload	χ^2	0 %*		0 %°		0 %*	
	D.F.	1	–	1	–	1	–
	χ^2	1 %***		0 %°		0 %***	

The explanatory variables that contribute the most significant explanatory power to a model (i.e., accounting for a large proportion of Wald χ^2) are shown in boldface

†: This explanatory variable is discarded during variable clustering analysis $|\rho| \geq 0.7$

–: Nonlinear degrees of freedom not allocated

Statistical significance of explanatory power according to Wald χ^2 likelihood ratio test:

° $p \geq 0.05$; * $p < 0.05$; ** $p < 1\%$; *** $p < 0.001$

variable because the number of reviewers of prior patches shares a more intuitive link with the response (i.e., the likelihood that a patch will be selected by reviewers) than the discussion length of prior patches.

After we remove the highly correlated explanatory variables, we repeat the variables clustering analysis and find that the number of files and churn are also highly correlated, i.e., Spearman's $|\rho| > 0.7$. Hence, we select the churn and remove the number of files because the distribution of churn data is less skewed than the number of files. For the Qt and OpenStack datasets, we obtain similar results from correlation analysis. Table 5 shows the results of our correlation analysis where the variables that were removed during the analysis are marked with a dagger symbol (†).

For the surviving explanatory variables, we perform a redundancy analysis to detect and remove redundant variables. We find that there are no explanatory variables that have a fit with an R^2 greater than 0.9. Hence, we use all of the surviving explanatory variables to construct our models.

(MC-2) Degree of Freedom Allocation We allocate the budgeted degrees of freedom to the surviving explanatory variables based on their potential for sharing a nonlinear relationship with the response variable. For example, Fig. 5 shows the potential for nonlinearity in the relationship between explanatory variables and the response variable in the Android dataset. We allocate additional degrees of freedom to the explanatory variables with a large Spearman multiple ρ^2.

By observing the rough clustering of variables according to the Spearman multiple ρ^2 values, we split the explanatory variables of Fig. 5 into three groups. We allocate: (1) five degrees of freedom to the number of reviewers of prior patches, (2) three degrees of freedom to feedback delay of prior patches, and (3) one degree of freedom to the remaining variables. We repeat the same process for the Qt and OpenStack datasets.

We then build our logistic regression models to fit our patch data using the surviving explanatory variables with the allocated degrees of freedom. Table 5 shows that the number

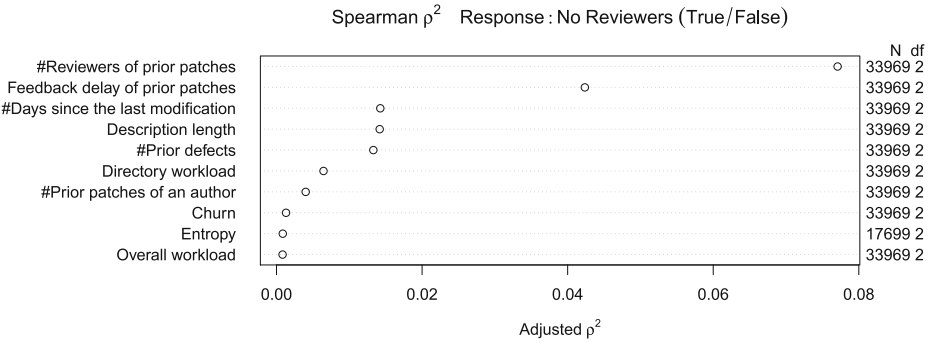


Fig. 5 Dotplot of the Spearman multiple ρ^2 of each explanatory variable and the response (the likelihood that a patch will not attract reviewers) in the Android dataset. Larger values indicate a higher potential for a nonlinear relationship (RQ1)

of degrees of freedom that we spent to fit our models did not exceed the budgeted degrees of freedom.

4.2 (RQ1-b) Model Analysis

In this section, we present and discuss the results of our model analysis approach that is outlined in Fig. 3 and present our empirical observations.

(MA-1) Assessment of Explanatory Ability & Model Reliability Table 5 shows that our models achieve an AUC of 0.7 to 0.74. Moreover, Table 5 also shows that the optimism of AUC is very small ($|\text{Optimism}| = 0.002$ for the Android project and $|\text{Optimism}| = 0.001$ for the Qt and OpenStack projects). These results indicate that our models are stable and can provide a meaningful and robust amount of explanatory power.

(MA-2) Power of Explanatory Variables Estimation Table 5 shows the explanatory power (Wald χ^2) of our explanatory variables that contribute to the fit of our models. In the table, the *Overall* column shows the proportion of the Wald χ^2 of the entire model fit that is attributed to that explanatory variable, and the *Nonlinear* column shows the proportion of the Wald χ^2 of the entire model fit that is attributed to the nonlinear component of that explanatory variable. The larger the proportion of the Wald χ^2 is, the larger the explanatory power that a particular explanatory contributes explanatory power to the fit of the model.

Table 5 shows that the number of reviewers of prior patches and the number of days since the last modification account for the largest proportion of Wald χ^2 in our three models. Hence, the number of reviewers of prior patches and the number of days since the last modification contribute the most significant explanatory power to the fit of our models. The description length also contributes a relatively large, significant amount of explanatory power to the Qt model.

On the other hand, Table 5 shows that churn did not contribute a significant amount of explanatory power to our three models. Moreover, we observe that entropy, the number of prior defects, feedback delay of prior patches, and the explanatory variables in the past involvement of an author and the review environment dimensions contribute a small explanatory power, although they have a statistically significant impact on our models.

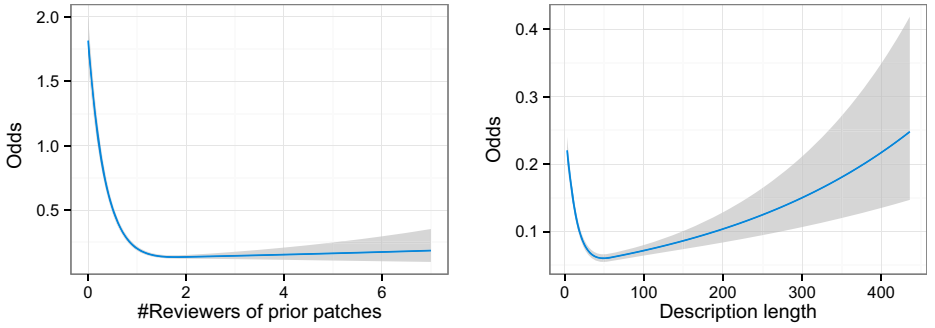
Table 5 also shows that four of the seven explanatory variables to which we allocated nonlinear degrees of freedom provide significant boosts to the explanatory power of the model. This result indicates that the nonlinear style of modelling is improving the fit of our models, providing a more in-depth picture of the relationship between explanatory variables and the response.

(MA-3) Examination of Variables in Relation to Response To study the relationship between the explanatory variables and the response, we plot the odds produced by our models against an explanatory variable while holding the other explanatory variables at their median values. For example, Fig. 6 shows the nonlinear relationship between the explanatory variables and the response with the 95 % confidence interval (gray area) based on models fit to 1,000 bootstrap samples.

To estimate the partial effect that the explanatory variables have on the likelihood, we analyze the relative change in the odds corresponding to a shift in the value of each explanatory variable. Table 6 shows the estimated partial effect of each explanatory variable in our models. The *Odds Ratio* column shows the partial effect based on the shifted value shown in the *Shifted Value* column. For continuous variables, the shifted value is an inter-quartile range of those explanatory variables. For categorical variables, the shifted value is a comparison between the observed category and the mode (i.e., the most frequently occurring category).

Below, we present and discuss our empirical observations from the examination of these explanatory variables in relation to the response.

Observation 1 - The number of reviewers of prior patches shares an inverse relationship with the likelihood that a patch will not be selected by reviewer Table 5 shows that there is a nonlinear relationship between the number of reviewers of prior patches and the likelihood that a patch will not be selected by reviewers. For example, Fig. 6a shows that the likelihood that an Android patch will not attract a reviewer decreases rapidly as the



(a) Number of reviewers of prior patches in the Android model

(b) Description length in the Qt model

Fig. 6 The nonlinear relationship between the likelihood that a patch will not attract reviewers (*y-axis*) and the explanatory variables (*x-axis*). The larger the odds value is, the higher the likelihood that the patch will not be selected by reviewers. The gray area shows the 95 % confidence interval estimated by using a bootstrap-derived approach

Table 6 Partial effect that our explanatory variables have on the likelihood that a patch will not attract reviewers (RQ1)

	Android		Qt		OpenStack	
	Shifted Value	Odds Ratio	Shifted Value	Odds Ratio	Shifted Value	Odds Ratio
Patch Properties Dimension						
Churn	5→96	0 %	4→63	0 %	5→90	0 %
Description length	11→45	-3 %↓	10→36	-57 %↓	13→47	-41 %↓
Entropy	1→1	0 %	1→1	-11 %↓	1→1	-11 %↓
Purpose	Feature→BUG-FIX	-10 %↓	Feature→BUG-FIX	0 %	Feature→BUG-FIX	-23 %↓
	Feature→Document	-37 %↓	Feature→Document	-30 %↓	Feature→Document	-25 %↓
History Dimension						
#Days since the last modification	1→109	-50 %↓	1→49	-45 %↓	1→23	-19 %↓
#Total authors	–		0→4	2%↑	–	
#Prior defects	0→4	-1 %↓	0→4	0 %	0→10	2 %↑
#Reviewers of prior patches	0→1	-89 %↓	1→2	-14 %↓	2→4	-40 %↓
Feedback delay of prior patches	0→3	9 %↑	0→8	0 %	1→6	16 %↑
Past Involvement of an Author Dimension						
#Prior patches of an author	3→105	-4 %↓	13→168	-6 %↓	4→114	4 %↑
Review Environment Dimension						
Overall workload	221→477	9 %↑	511→674	-5 %↓	1220→1626	-6 %↓
Directory workload	1→10	3 %↑	1→12	1 %↑	2→33	2 %↑

The larger the magnitude of the odds ratio is, the larger the partial effect that an explanatory variable has on the likelihood

number of reviewers of prior patches increases from 0 to 1. We observe similar trends in the Qt and OpenStack models. Furthermore, Table 6 shows that the likelihood can decrease by 89 %, 14 %, and 40 % when the number of reviewers of prior patches increases from 0 to 1, 1 to 2, and 2 to 4 in the Android, Qt, OpenStack models, respectively. Broadly speaking, our models show that patches are more likely to attract reviewers if past changes to the modified files have a tendency to be reviewed by at least two reviewers. These results indicate that patches that modify files whose prior patches have had few reviewers tend to be ignored by reviewers.

Observation 2 - The number of days since the last modification shares an inverse relationship with the likelihood that a patch will not be selected by reviewers Table 6 also shows that the number of days since the last modification consistently shares a strong inverse relationship with the likelihood that a patch will be not selected by reviewers in the three models for the studied projects. The likelihood decreases by 50 %, 45 %, and 19 % when the number of days since the last modification is changed from 1 to 109, 1 to 49, and 1 to 23 in the Android, Qt, and OpenStack models, respectively. This result indicates that a patch containing files that have been recently modified is not likely to attract reviewers.

Observation 3 - Description length shares an inverse relationship with the likelihood that a patch will not be selected by reviewers Figure 6b shows that there is a decreasing trend in the likelihood that a Qt patch will not attract reviewers as the description length increases. On the other hand, there is an increasing trend in the likelihood as the description length increases beyond 50. However, the broadening of the confidence interval (gray area) indicates that there is less data to support this area of the curve. Table 6 also shows that the description length shares a relatively strong relationship with the likelihood in the Qt and OpenStack models. When the description length is greater than 10 words, the likelihood decreases by 3 %, 57 %, and 41 % in the Android, Qt, and OpenStack models, respectively. This result indicates that a patch with a short description is unlikely to attract reviewers.

The number of reviewers of prior patches and the number of days since the last modification share a strong increasing relationship with the likelihood that a patch will have at least one reviewers. Furthermore, a short patch description can also lower the likelihood of attracting reviewers (Observations 1-3).

(RQ2) What patch characteristics share a relationship with the likelihood of a patch not being discussed?

A review that simply assigns a review score without any suggestion for improvement nor discussion provides little return on code review investment. Hence, to address our RQ2, we perform our analysis on patches that have reviewers but were not discussed. We filter the patches that did not attract any reviewers, since such patches cannot receive any feedback. We then identify the patches that are not discussed by counting the number of messages that are posted in the review discussion thread of each patch. We classify the patches that had no messages as *patches that are not discussed*. Patches that had at least one message are defined as patches that are discussed.

Table 7 Patch data for the study of RQ2

Project	Patch data		
	Studied period	#Patches that are not discussed (TRUE class)	#Patches with reviewers
Android	2012/01 - 2014/12	12,262	25,613
Qt	2012/01 - 2014/12	29,771	58,894
OpenStack	2013/01 - 2014/12	21,855	87,255

Table 7 provides an overview of the studied patch data after we remove patches that did not attract reviewers. Below, we present and discuss the results of our model construction and analysis.

4.3 (RQ2-a) Model Construction

Similar to RQ1, the response variable is set to TRUE if a patch is not discussed and FALSE otherwise. We then use all of our patch and MCR process metrics that are described in Table 2 to construct logistic regression models for each studied project.

(MC-1) Correlation & Redundancy Analysis Since we filtered out patches that did not attract any reviewers and we added the past involvement of reviewers metrics into the models, we have to perform correlation & redundancy analysis for the studied patch data again. Figure 7 shows that there are four clusters of variables that have a Spearman's $|\rho| > 0.7$: (1) the variables in the past involvement of an author dimension, (2) the number of files and directories, (3) the number of prior patches of reviewers and the number of recent patches of reviewers, (4) the number of prior defects and the number of total authors, and (5) the number of reviewers and the discussion length of prior patches. For the first three clusters, we select the number of prior patches of an author, the number of files, and the number of prior patches of reviewers as the representative variables because they are simpler to calculate than the other variables. For the forth cluster, we select the number of prior defects as the representative variable since the distribution of the number of prior defects is

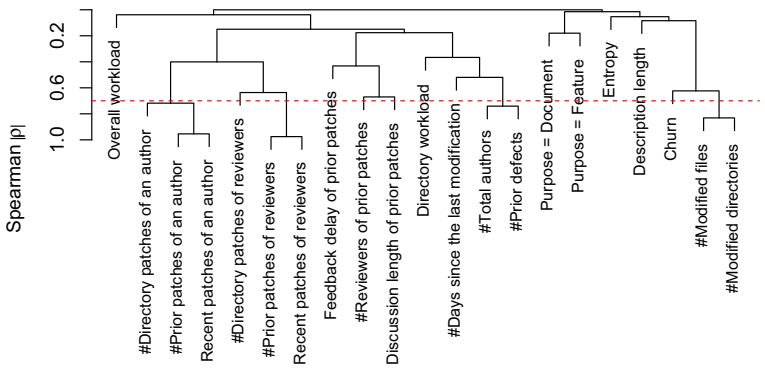


Fig. 7 Hierarchical clustering of variables according to Spearman's $|\rho|$ in the Android dataset (RQ2). The dashed line indicates the high correlation threshold (i.e., Spearman's $|\rho| = 0.7$)

Table 8 Statistics of the logistic regression models for identifying patches that are not discussed (RQ2)

		Android		Qt		OpenStack	
AUC (|Optimism|)		0.70 (0.002)		0.72 (0.001)		0.78 (0.001)	
Budgeted D.F.		817		1,941		1,457	
Spent D.F.		19		19		22	
Wald χ^2		1,312***		3,681***		5,882***	
		Overall	Nonlinear	Overall	Nonlinear	Overall	Nonlinear
Patch Properties Dimension							
Churn	D.F.	2	1	2	1	2	1
	χ^2	12 %***	12 %***	19 %***	19 %***	8 %***	8 %***
#Modified files	D.F.	†		†		†	
	χ^2						
#Modified directories	D.F.	†		†		†	
	χ^2						
Entropy	D.F.	1	–	2	1	1	–
	χ^2	0 %°		2 %***	2 %***	0 %***	
Description length	D.F.	4	3	2	1	4	3
	χ^2	18 %***	7 %***	5 %***	1 %***	17 %***	4 %***
Purpose	D.F.	2	–	2	–	2	–
	χ^2	0 %°		1 %***		0 %*	
History Dimension							
#Days since the last modification	D.F.	1	–	1	–	1	–
	χ^2	2 %***		%***		3 %***	
#Total authors	D.F.	†		†		†	
	χ^2						
#Prior defects	D.F.	2	1	1	–	2	1
	χ^2	8 %***	2 %***	1 %***		3 %***	2 %***
#Reviewers of prior patches	D.F.	†		†		†	
	χ^2						
Discussion length of prior patches	D.F.	2	1	2	1	2	1
	χ^2	14 %***	12 %***	23 %***	21 %***	27 %***	17 %***
Feedback delay of prior patches	D.F.	1	–	1	–	1	–
	χ^2	1 %**		0 %°		0 %°	
Past Involvement of an Author Dimension							
#Prior patches of an author	D.F.	1	–	2	1	1	–
	χ^2	30 %***		25 %***	4 %***	7 %***	
Recent patches of an author	D.F.	†		†		†	
	χ^2						
#Directory patches of an author	D.F.	†		†		†	
	χ^2						
Past Involvement of Reviewers Dimension							
#Prior patches of reviewers	D.F.	1	–	1	–	2	1
	χ^2	10 %***		14 %***		5 %***	0 %°

Table 8 (continued)

		Android		Qt		OpenStack	
AUC (|Optimism|)		0.70 (0.002)		0.72 (0.001)		0.78 (0.001)	
Budgeted D.F.		817		1,941		1,457	
Spent D.F.		19		19		22	
Wald χ^2		1,312***		3,681***		5,882***	
		Overall	Nonlinear	Overall	Nonlinear	Overall	Nonlinear
Recent patches of reviewers	D.F.	†		†		†	
	χ^2						
#Directory patches of reviewers	D.F.	†		1	–	2	1
	χ^2			1 %***		2 %***	1 %***
Review Environment Dimension							
Overall workload	D.F.	1	–	1	–	1	–
	χ^2	0 %*		0 %***		0 %***	
Directory workload	D.F.	1	–	1	–	1	–
	χ^2	0 %°		1 %***		1 %***	

The explanatory variables that contribute the most significant explanatory power to a model (i.e., accounting for a large proportion of Wald χ^2) are shown in boldface

†: This explanatory variable is discarded during variable clustering analysis $|\rho| \geq 0.7$

–: Nonlinear degrees of freedom not allocated;

Statistical significance of explanatory power according to Wald χ^2 likelihood ratio test:

$\circ p \geq 0.05$; * $p < 0.05$; ** $p < 1\%$; *** $p < 0.001$

less skewed than the total number of authors. For the fifth cluster, we select the discussion length of prior patches because it shares a more intuitive link with the response (i.e., the likelihood that a patch will not be discussed) than the number of reviewers of prior patches. For the Qt and OpenStack datasets, Table 8 shows the results of our correlation analysis.

(MC-2) Degree of Freedom Allocation Figure 8 shows the estimated potential for non-linear relationships between each explanatory variable and the likelihood that a patch will not be discussed in the Android dataset. We split the explanatory variables for the Android dataset into three groups: (1) the description length (2) churn, the number of prior defects, discussion length of prior patches, the number of prior patches of an author and the number of prior patches of reviewers, and (3) the remaining explanatory variables. We then allocate five degrees of freedom to the first group, three degrees of freedom to the second group, and one degree of freedom to the third group. We repeat the same process for the Qt and OpenStack datasets.

4.4 (RQ2-b) Model Analysis

In this section, we present the results of our model analysis approach that is outlined in Fig. 3 and present our empirical observations.

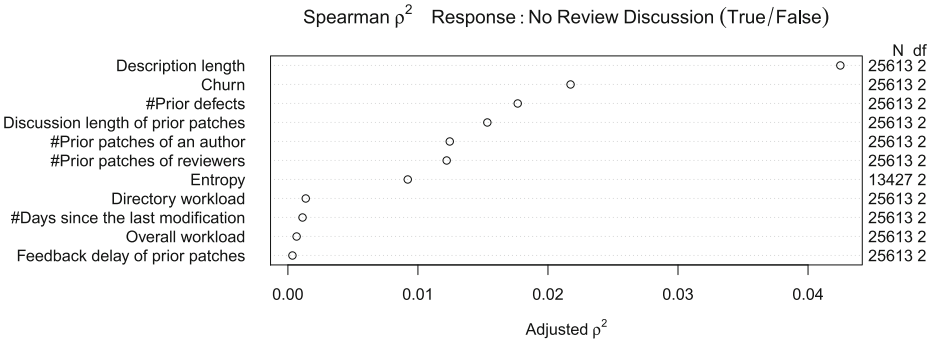


Fig. 8 Dotplot of the Spearman multiple ρ^2 of each explanatory variable and the response (the likelihood that a patch will not be discussed) in the Android dataset. Larger values indicate a higher potential for a nonlinear relationship (RQ2)

(MA-1) Assessment of Explanatory Ability & Model Reliability Table 8 shows that our models achieve an AUC of 0.70 to 0.78 and the optimism of AUC is very small for all of our studied datasets. These results indicate that our models can provide a meaningful and robust amount of explanatory power.

(MA-2) Power of Explanatory Variables Estimation Table 8 shows the proportion of Wald χ^2 of the explanatory variables that contribute to the fit of our models. We find that the discussion length of prior patches has a large proportion of Wald χ^2 in our three models. The churn, the description length, the number of prior patches of an author and reviewers also account for a large proportion of the explanatory power in two of the three models.

Table 8 shows that entropy, feedback delay of prior patches, the number of prior patches within the same directory that the reviewer has reviewed, and the explanatory variables in the review environment dimension did not contribute a significant amount of explanatory, although they survive our correlation and redundancy analysis. These results indicate that these explanatory variables in our models share a weaker relationship with the likelihood of a patch being discussed than other metrics.

Table 8 also shows that five of the fifteen explanatory variables to which we allocated nonlinear degrees of freedom provide significant boosts to the explanatory power of the model. This result indicates that the nonlinear style of modelling can improve the fit of our models. However, we find that the nonlinear degrees of freedom that we allocate to the number of prior patches of reviewers does not provide a significant amount of explanatory power to the OpenStack model, suggesting that not all relationships with potential for nonlinearity benefit from nonlinear fits.

(MA-3) Examination of Variables in Relation to Response Figure 9 shows the nonlinear relationships of the high impact explanatory variables and the response. We provide online access to the figures for the other studied projects.[8] Table 9 shows the estimated partial effect of that each explanatory variable has on the likelihood that a patch will not be discussed. Below, we present and discuss our empirical observations from the examination of these explanatory variables in relation to the response.

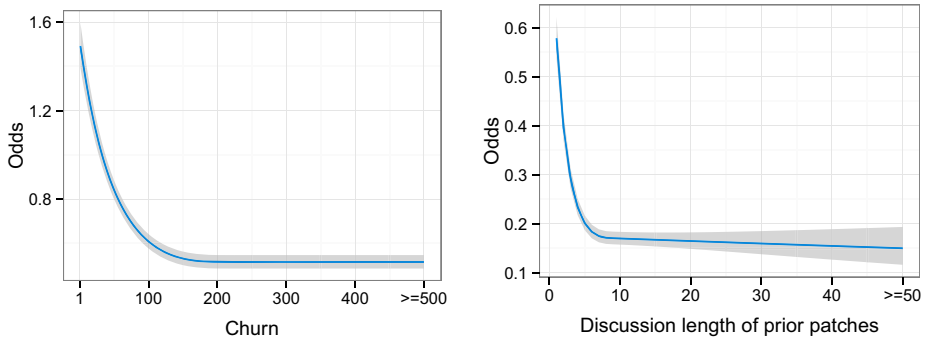
[8] http://sailhome.cs.queensu.ca/replication/review_participation/

Observation 4 - Churn shares an inverse relationship with the likelihood that a patch will not be discussed We observe that churn shares a strong relationship with the likelihood that a patch will not be discussed in our three models. Table 9 shows that the likelihood decreases by 30 %, 46 %, and 40 % in the Android, Qt, OpenStack models, respectively. Figure 9a shows that there is a decreasing trend of the odds in the Qt model when the churn increases from 1 to 150 LOC. Then, the odds stabilizes when the churn increases to more than 150 LOC. We also observe similar trend of the odds produced by the Android model. The odds decrease when the churn increases from 1 to 300 LOC, then the odds stabilize when the churn increases to more than 300 LOC. This result indicates that the more lines that were changed in the patch, the more likely the patch will be discussed.

Observation 5 - The description length shares an inverse relationship with the likelihood that a patch will not be discussed Table 9 shows that the description length consistently shares an inverse relationship with the likelihood that a patch will not be discussed in our three studied projects. The likelihood decreases by 43 %, 24 %, and 43 % when the description length is greater than 11 words. Our results suggest that the longer the description that an author provides, the higher the likelihood of the patch being discussed.

Observation 6 - The discussion length of prior patches shares an inverse relationship with the likelihood that a patch will not be discussed Figure 9b shows that the odds, produced by the OpenStack model, sharply decrease when the discussion length of prior patches increases from 0 to 10 messages. We observe similar trends of the odds in the Android and Qt models. Table 9 shows that the likelihood decreases by 45 %, 58 %, and 67 % when the discussion length increases beyond 2 messages in the Android, Qt, OpenStack models, respectively. These results indicate that patches that modify files whose prior patches typically had short review discussions are unlikely to be discussed.

Observation 7 - The number of prior patches of an author shares an increasing relationship with the likelihood that a patch will not be discussed, while the number of prior patches of reviewers shares an inverse relationship with the likelihood



(a) Churn in the Qt model

(b) Discussion length of prior patches in the OpenStack model

Fig. 9 The nonlinear relationship between the likelihood that a patch will not be discussed (*y-axis*) and the explanatory variables (*x-axis*). The larger the odds value is, the higher the likelihood that the patch will not be not discussed. The gray area shows the 95 % confidence interval estimated by using a bootstrap-derived approach

Table 9 Partial effect that our explanatory variables have on the likelihood that a patch will not be discussed (RQ2)

	Android		Qt		OpenStack	
	Shifted Value	Odds Ratio	Shifted Value	Odds Ratio	Shifted Value	Odds Ratio
Patch Properties Dimension						
Churn	5→96	-30 %↓	4→60	-46 %↓	5→91	-40 %↓
Description length	11→46	-43 %↓	11→38	-24 %↓	15→48	-43 %↓
Entropy	1→1	-3 %↓	1→1	6 %↑	1→1	7 %↑
Purpose	Feature→BUG-FIX	-1 %↓	Feature→BUG-FIX	-19 %↓	Feature→BUG-FIX	-7 %↓
	Feature→Document	7 %↑	Feature→Document	-11 %↓	Feature→Document	-9 %↓
History Dimension						
#Days since the last modification	2→92	-16 %↓	1→52	-7 %↓	1→22	-8 %↓
#Prior defects	0→5	-27 %↓	0→4	-3 %↓	1→11	-34 %↓
Discussion length of prior patches	0→2	-45 %↓	0→2	-58 %↓	1→6	-67 %↓
Feedback delay of prior patches	0→4	0 %	0→8	0 %	1→6	0 %
Past Involvement of an Author Dimension						
#Prior patches of an author	3→111	41 %↑	12→167	104 %↑	4→114	21 %↑
Past Involvement of Reviewers Dimension						
#Prior patches of reviewers	11→207	-26 %↓	31→278	-32 %↓	56→532	-37 %↓
#Directory patches of reviewers	–		3→107	4 %↑	16→183	-18 %↓
Review Environment Dimension						
Overall workload	221→476	-7 %↓	507→673	-6 %↓	1225→1626	10 %↑
Directory workload	1→10	0 %	1→12	6 %↑	2→31	-3 %↓

The larger the magnitude of the odds ratio is, the larger the partial effect that an explanatory variable has on the likelihood

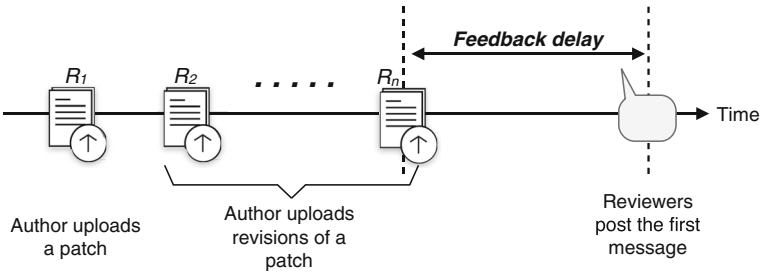


Fig. 10 An example of a calculation for feedback delay

Table 9 shows that the likelihood increases by 41 %, 104 %, and 21 % when the number of prior patches of an author is greater than 3, 12, and 4 in the Android, Qt, OpenStack models, respectively. Furthermore, Table 9 shows that the number of prior patches of reviewers shares an inverse relationship with the likelihood. The likelihood can decrease by 26 %, 32 %, and 37 % in the Android, Qt, and OpenStack models, respectively. Our results indicate that a patch written by an experienced developer or examined by an inexperienced reviewer is not likely to be discussed.

Churn, the description length, and the discussion length of prior patches share an increasing relationship with the likelihood that a patch will be discussed. Moreover, the past involvement of an author shares an inverse relationship, while the past involvement of reviewers shares an increasing relationship with the likelihood (Observations 4-7).

(RQ3) What patch characteristics share a relationship with the likelihood of a patch receiving slow initial feedback?

To address our RQ3, we identify patches that receive slow initial feedback by measuring feedback delay, i.e., the time between the submission of the latest patch revision before the first message from reviewers is posted until that message is posted. Figure 10 provides an example of a calculation for feedback delay. We did not use the time that the original patch is submitted because there are likely cases that reviewers are still waiting for the author to complete preliminary revisions, which would incorrectly inflate the feedback delay.

To follow our model construction approach, we classify the patches into two groups, i.e., patches that receive prompt initial feedback and patches that receive slow initial feedback. From the descriptive statistics of feedback delay (see Table 10), we classify the patches that have a feedback delay of more than 12 hours as *patches that receive slow initial feedback*. Patches that receive initial feedback within 12 hours are defined as patches that receive prompt initial feedback.

Table 10 Descriptive statistics of feedback delay (hours)

Project	Min	1st Qu.	Median	Mean	3rd Qu.	Max
Android	0.00	0.19	1.20	106.50	11.78	16,470.00
Qt	0.00	0.28	1.49	70.75	15.70	20,260.00
OpenStack	0.00	0.27	2.17	31.82	15.17	7,031.00

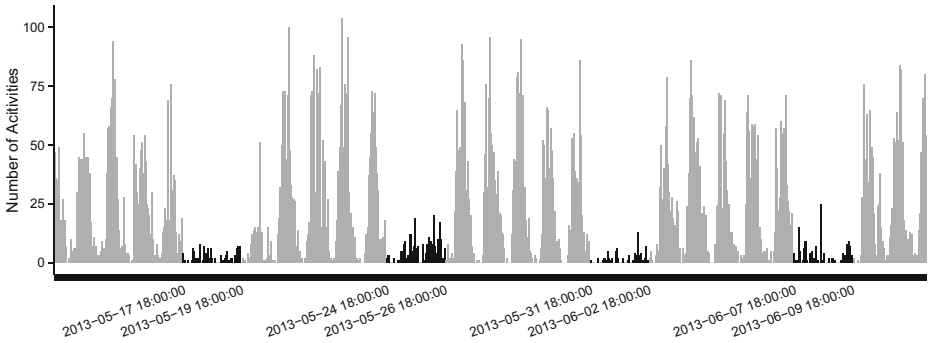


Fig. 11 The hourly code review activity of the Qt project. The dark areas indicate the periods that are likely to be weekends

Similar to RQ2, we want to investigate the characteristics of patches that receive slow initial feedback although these patches eventually have reviewers providing feedback. Hence, we filter out patches that did not attract any reviewers, since such patches do not receive any feedback. Moreover, we filter out patches that are submitted on weekends, since our prior study finds that code review activity is often less active on weekend than on weekdays (Thongtanunam et al. 2014). Hence, such patches have a lower chance to receive initial feedback within 12 hours. Since our studied projects have globally-distributed development teams, the timezone difference could make it difficult to detect weekends consistently. To identify weekends, we observe the number of code review activities (i.e., a patch submission, posting a review message, or voting a review score). For example, Fig. 11 shows the hourly code review activity of the Qt project, where the periods with small amounts of code review activities are indicated using dark shading. We mark the areas with small amounts of code review activities as weekends for the Qt project, and we remove the patches that are submitted in these periods from our analysis. We repeat the same process for the Android and OpenStack projects.

Table 11 provides an overview of the studied patch data after we remove patches that did not attract reviewers and that are submitted during weekends. Below, we present and discuss the results of our model construction and analysis.

4.5 (RQ3-b) Model Construction

Again, we use our patch and MCR process metrics that are described in Table 2 as explanatory variables. The response is set to TRUE if a patch receives slow initial feedback, and

Table 11 Patch data for the study of RQ3

Project	Patch data		
	Studied period	#Patches that received slow initial feedback (TRUE class)	#Patches with reviewers & submitted in workdays
Android	2012/01 - 2014/12	5,759	23,287
Qt	2012/01 - 2014/12	15,900	54,783
OpenStack	2013/01 - 2014/12	22,302	79,431

Table 12 Statistics of the logistic regression models for identifying patches receiving slow initial feedback (RQ3)

		Android		Qt		OpenStack	
AUC (|Optimism|)		0.66 (0.004)		0.61 (0.002)		0.61 (0.001)	
Budgeted D.F.		383		1,060		1,486	
Spent D.F.		18		17		16	
Wald χ^2		722***		940***		1,380***	
		Overall	Nonlinear	Overall	Nonlinear	Overall	Nonlinear
Patch Properties Dimension							
Churn	D.F.	1	–	2	1	2	1
	χ^2	0 %°		6 %***	6 %***	15 %***	15 %***
#Modified files	D.F.	†		†		†	
	χ^2						
#Modified directories	D.F.	†		†		†	
	χ^2						
Entropy	D.F.	1	–	2	1	1	–
	χ^2	2 %***		7 %***	4 %***	1 %**	
Description length	D.F.	1	–	1	–	1	–
	χ^2	8 %***		0 %°		1 %***	
Purpose	D.F.	2	–	2	–	2	–
	χ^2	8 %***		4 %***		12 %***	
History Dimension							
#Days since the last modification	D.F.	2	1	1	–	1	–
	χ^2	7 %***	1 %*	4 %***		13 %***	
#Total authors	D.F.	†		†		†	
	χ^2						
#Prior defects	D.F.	1	–	1	–	1	–
	χ^2	1 %°		0 %°		1 %***	
#Reviewers of prior patches	D.F.	†			†		†
	χ^2						
Discussion length of prior patches	D.F.	1	–	1	–	1	–
	χ^2	1 %**		1 %*		0 %°	
Feedback delay of prior patches	D.F.	2	1	2	1	2	1
	χ^2	33 %***	27 %***	51 %***	51 %***	61 %***	54 %***
Past Involvement of an Author Dimension							
#Prior patches of an author	D.F.	4	3	1	–	1	–
	χ^2	18 %***	7 %***	5 %***		0 %*	
Recent patches of an author	D.F.	†		†		†	
	χ^2						
#Directory patches of an author	D.F.	†		†		†	
	χ^2						
Past Involvement of Reviewers Dimension							
#Prior patches of reviewers	D.F.	1	–	1	–	1	–
	χ^2	0 %°		6 %***		0 %°	

Table 12 (continued)

		Android		Qt		OpenStack	
AUC (|Optimism|)		0.66 (0.004)		0.61 (0.002)		0.61 (0.001)	
Budgeted D.F.		383		1,060		1,486	
Spent D.F.		18		17		16	
Wald χ^2		722***		940***		1,380***	
		Overall	Nonlinear	Overall	Nonlinear	Overall	Nonlinear
Recent patches of reviewers	D.F.	†		†		†	
	χ^2						
#Directory patches of reviewers	D.F.	†		1	–	1	–
	χ^2			0 %°		3 %***	
Review Environment Dimension							
Overall workload	D.F.	1	–	1	–	1	–
	χ^2	2 %***		5 %***		0 %°	
Directory workload	D.F.	1	–	1	–	1	–
	χ^2	0 %°		5 %***		0 %°	

The explanatory variables that contribute the most significant explanatory power to a model (i.e., accounting for a large proportion of Wald χ^2) are shown in boldface

†: This explanatory variable is discarded during variable clustering analysis $|\rho| \geq 0.7$

–: Nonlinear degrees of freedom not allocated

Statistical significance of explanatory power according to Wald χ^2 likelihood ratio test:

$\circ p \geq 0.05$; * $p < 0.05$; ** $p < 1\%$; *** $p < 0.001$

FALSE otherwise. We then construct a logistic regression model, which we describe in detail below.

(MC-1) Correlation & Redundancy Analysis We check for highly correlated and redundant variables again, since we filter out patches that did not attract any reviewers, and were submitted on weekends. We find that the hierarchical clustering analysis shows the same results as the analysis in RQ2. Hence, we use the same set of surviving variables. Table 12 shows the results of our correlation analysis.

(MC-2) Degree of Freedom Allocation Again, we allocate additional degrees of freedom to the surviving explanatory variables based on their potential for sharing a nonlinear relationship with the response. By observing the Spearman multiple ρ^2 values in Fig. 12, we split the explanatory variables into two groups: (1) the feedback delay of prior patches and the number of prior patches of an author, and (2) the remaining explanatory variables. We then allocate five and one degree of freedom to each group respectively. We repeat the same process for Qt and OpenStack datasets. Table 12 shows the number of degrees of freedom that we spent to build our logistic regression models.

4.6 (RQ3-c) Model Analysis

In this section, we describe the results of our model analysis approach that is outlined in Fig. 3 and present our empirical observations.

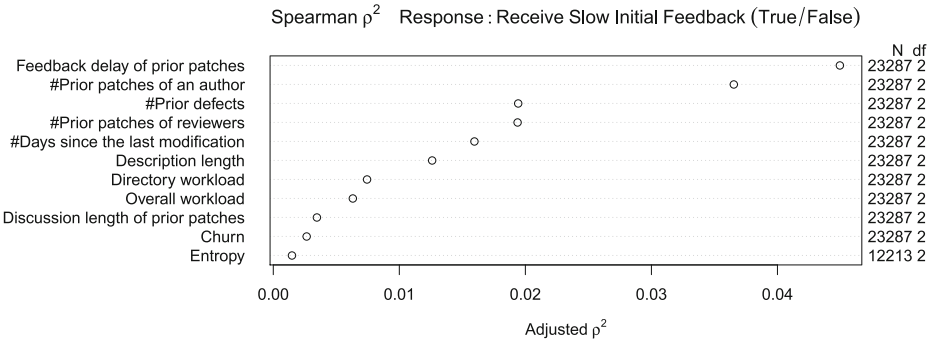


Fig. 12 Dotplot of the Spearman multiple ρ^2 of each explanatory variable and the response (the likelihood that a patch will receive slow initial feedback) in the Android model. Larger values indicate a higher potential for a nonlinear relationship (RQ3)

(MA-1) Assessment of Explanatory Ability & Model Reliability Table 12 shows that our models achieve an AUC of 0.61 to 0.66 and the optimism of 0.001 (OpenStack)-0.004 (Android). These results indicate that our models can determine the likelihood that a patch will receive slow initial feedback and provide a meaningful and robust amount of explanatory power.

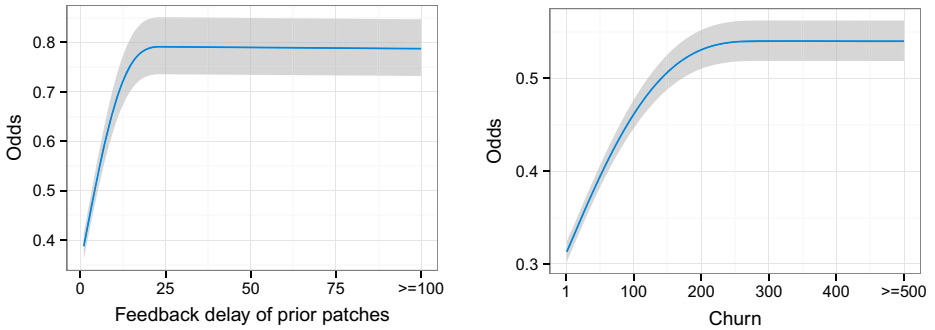
(MA-2) Power of Explanatory Variables Estimation We find that the feedback delay of prior patches contribute a significant amount of explanatory power to the fit of our models. Table 12 shows that feedback delay of prior patches accounts for a large proportion of the explanatory power in our three models. Furthermore, we observe that churn also accounts for a large proportion of the explanatory power of the OpenStack model, while the number of prior patches of an author accounts for a large proportion of the explanatory power of the Android model.

Table 12 shows that the explanatory power of the variables in the past involvement of reviewers dimension account for a relatively small proportion of the explanatory power. This result suggests that when considering the initial feedback delay, the experience of reviewers is not as important as the other studied dimensions.

(MA-3) Examination of Variables in Relation to Response Figure 13 shows the shape of the nonlinear relationship between the likelihood that a patch will receive slow initial feedback and two of the most impactful explanatory variables. For the other studied projects, we provide online access to the figures.[9] Table 13 shows the estimated partial effect that each explanatory variable has on the likelihood. Below, we present our observations from the examination of these explanatory variables in relation to the response variable.

Observation 8 - The feedback delay of prior patches has an increasing relationship with the likelihood that a patch will receive slow initial feedback Figure 13a shows that the shape of the relationship between the likelihood and the feedback delay of prior patches in the Qt model. The plot shows a steeply increasing trend in the likelihood where

[9] http://sailhome.cs.queensu.ca/replication/review_participation/



(a) Feedback delay of prior patches in the Qt model

(b) Churn in the OpenStack model

Fig. 13 The nonlinear relationship between the likelihood that a patch will receive slow initial feedback (*y-axis*) and the explanatory variables (*x-axis*). The larger the odds value is, the higher the likelihood that the patch will receive slow initial feedback. The gray area shows the 95 % confidence interval estimated by using a bootstrap-derived approach

the feedback delay of prior patches reaches to 25 hours. We observe the similar trends in the Android and OpenStack models. Table 13 also shows that feedback delay of prior patches shares a strong relationship with the likelihood. When the feedback delay of prior patches is greater than one hour, the likelihood increases by 35 %, 69 %, and 76 % in the Android, Qt, and OpenStack models, respectively. These results indicate that patches that modify files whose prior patches had received slow initial feedback tend to also receive slow initial feedback.

We further investigate how long the association between the feedback delay of prior and future patches lasts. To do so, we compute the autocorrelation of the feedback delay. First, we classify patches that receive slow initial feedback and patches that receive prompt initial feedback. Then, for each file, we create a series of patches that impact that file. Finally, we measure the association of receiving slow initial feedback of each series with that series itself using Cramér's V (Cramėr 1999). We also use the convention of Rea and Parker (2014) for describing the magnitude of an association. Figure 14 shows the correlogram where the x-axis indicates the sample intervals of lags (i.e., patches) and y-axis indicates the median Cramér's V values between the series of patches at sequence t and the series at sequence t+lag for all studied files. The correlation at the lag value of 0 always equals to 1. Figure 14 shows that the median of autocorrelation is moderate ($0.20 \leq V < 0.40$) until the lag is 11, 12, and 2 for Android, Qt, and OpenStack, respectively. Then, the median of autocorrelation is weak or negligible ($V < 0.2$). These results indicate that receiving slow initial feedback in the current patch is associated with receiving slow initial feedback in the next 11 (Android), 12(Qt), and 2(OpenStack) patches.

Observation 9 - The purpose of introducing a new feature shares an increasing relationship with the likelihood that a patch will receive slow initial feedback Table 13 shows that a patch will have a lower likelihood of receiving slow initial feedback when its purpose is changed from feature introduction to other purposes. The likelihood decreases by 16 % to 29 % when the purpose of the patch is changed from feature introduction to bug fixing, and 12 % to 40 % when the purpose is changed to documentation. This result indicates that a patch that introduces new functionality is more likely to receive slow initial feedback than patches for other purposes.

Table 13 Partial effect that our explanatory variables have on the likelihood that a patch will receive slow initial feedback (RQ3)

	Android		Qt		OpenStack	
	Shifted Value	Odds Ratio	Shifted Value	Odds Ratio	Shifted Value	Odds Ratio
Patch Properties Dimension						
Churn	5→96	0 %	4→59	19 %↑	5→90	33 %↑
Entropy	1→1	-9 %↓	1→1	-12 %↓	1→1	4 %↑
Description length	11→46	12 %↑	11→37	2 %↑	15→48	4 %↑
Purpose	Feature→BUG-FIX	-29 %↓	Feature→BUG-FIX	-16 %↓	Feature→BUG-FIX	-25 %↓
	Feature→Document	-40 %↓	Feature→Document	-12 %↓	Feature→Document	-22 %↓
History Dimension						
#Days since the last modification	2→93	12 %↑	1→51	8 %↑	1→22	7 %↑
#Prior defects	0→5	8 %↑	0→4	0 %	1→11	2 %↑
Discussion length of prior patches	0→2	-3 %↓	0→2	-1 %↓	1→6	0 %
Feedback delay of prior patches	0→4	35 %↑	0→8	69 %↑	1→6	76 %↑
Past Involvement of an Author Dimension						
#Prior patches of an author	3→109	-42 %↓	12→170	9 %↑	4→111	-2 %↓
Past Involvement of Reviewers Dimension						
#Prior patches of reviewers	11→208	-2 %↓	32→275	-11 %↓	56→527	1 %↑
#Directory patches of reviewers			3→106	0 %	15→183	-4 %↓
Review Environment Dimension						
Overall workload	221→476	-12 %↓	507→674	-11 %↓	1229→1626	0 %
Directory workload	0→10	-1 %↓	1→12	-6 %↓	2→31	0 %

The larger the magnitude of the odds ratio is, the larger the partial effect that an explanatory variable has on the likelihood

In addition, we also find that churn shares an increasing relationship with the likelihood that a patch will receive slow initial feedback. Figure 13b shows that the odds, produced by the OpenStack model, increases when churn increases from 1 to 150 LOC. Table 13 shows that when the churn increases greater than 5 LOC, the likelihood increases by 33 % in the OpenStack models. However, Table 12 shows that churn did not contribute a significant amount of explanatory power to the Android and Qt models. Moreover, Table 13 shows that the partial effect of churn having on the likelihood in the Android model is 0 %. To better understand this relationship, we further investigate with a one-tailed Mann-Whitney U test ($\alpha = 0.05$) comparing the churn of patches that receive slow initial feedback and those that receive prompt feedback in the Android dataset. We find that the churn of patches that receive slow initial feedback is statistically larger than churn of patches that receive slow initial feedback, indicating that a patch with large churn tends to receive slow initial feedback (p-value < 0.001). Our results suggest that when we control for several confounding factors using prediction model, the churn did not share a strong relationship with the likelihood.

On the other hand, we find that the number of prior patches of an author shares a strong relationship with the speed of initial feedback in the Android model. Table 12 shows that the number of prior patches of an author contributes a large amount of explanatory power, and its nonlinear relationship can boost to the explanatory power of the Android model. Table 13 shows that the likelihood decreases by 42 % when the number of prior patches of an author increases from 3 to 109.

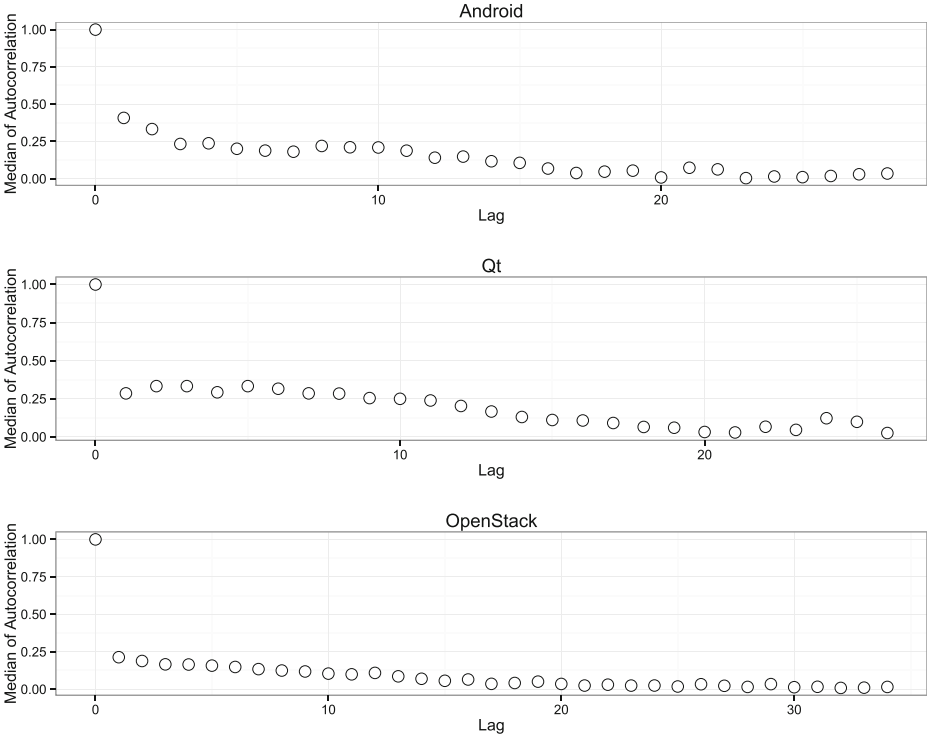


Fig. 14 Correlogram of autocorrelation of feedback delay of prior patches

We observe a similar relationship in the OpenStack dataset, i.e., the likelihood decrease by 4 % when the number of prior patches of an author increases from 4 to 111. However, Table 13 shows that the likelihood increases when the number of prior patches of an author in the Qt dataset increases. One possible reason for the reverse effect of the number of prior patches of an author in the Qt dataset is the low number of developers in the sub-projects. This is especially true for the `qt3d` sub-project. We find that 52 % of the Qt patches that are submitted to the `qt3d` sub-project received slow initial feedback (609/1,166). However, there are only 24 developers who author these patches. Hence, many of the `qt3d` patches have a large number of prior patches of an author while they received slow initial feedback.

The feedback delay of prior patches shares a strong relationship with the likelihood that a patch will receive slow initial feedback. Furthermore, the purpose of introducing new features can also increase the likelihood (Observations 8-9).

5 Discussion

In this section, we provide a broader discussion of our empirical observations. The observations are grouped into three main groups, which are the (1) past review participation, (2) past activities of practitioners, and (3) patch properties.

5.1 Past Review Participation

The Number of Reviewers of Prior Patches As we conjecture (see Table 2), observation 1 shows that the number of reviewers of prior patches share an increasing relationship the likelihood of having at least one reviewer. We suspect that files have had few reviewers of prior patches in part due to a limited number of developers who are interested or working on related subsystems. Prior work reports that developers usually select patches that are within their area of interest (Rigby and Storey 2011). For example, we find that 42 % of the patches in the less popular subsystems (i.e., having fewer than 10 contributing developers) of Android contain files where their prior patches do not attract any reviewers. We also observe a similar proportion of 45 % and 53 % in the small subsystems of Qt and OpenStack. On the other hand, the subsystems that have a large number of developers (11 to 590 contributing developers) have only 8 %, 3 %, and 6 % of patches containing files where their prior patches did not attract any reviewers in the Android, Qt, and OpenStack projects, respectively.

Moreover, our results indicate that patches are more likely to have at least one reviewer if past changes to the modified files have a tendency to be reviewed by at least two reviewers. This finding complements those of Rigby and Bird (2013) who report that two reviewers find an optimal number of defects in MCR processes.

Discussion Length of Prior Patches Observation 6 arrives at our conjecture where the discussion length shares an increasing relationship with the likelihood that a patch will be discussed. In other words, files that had little review discussion in the past will have little discussion in the future. Similar to prior studies, the discussion length is a strong indicator for the review quality. For example, Kononenko et al. (2015) report that the number of review comments posted shares a significant link to the defect-proneness of that patch. Our

prior work also shows that files without post-release defects also undergo many reviews with long discussions (Thongtanunam et al. 2015a). In addition, this observation also complements the study of Bird et al. (2006) who find that the social status shares a strong relationship with the number of messages replied in the email-based discussions.

During correlation analysis, we find that the number of reviewers of prior patches and the discussion length of prior patches are highly correlated with each other. Hence, we experimented by swapping these variables in order to validate observations 1 and 6. We find that the number of reviewers of prior patches and the discussion length of prior patches share a similar relationship with the likelihood. Therefore, for observation 1, we can conclude that patches that modify files whose prior patches had received either few reviewers or short discussions tend tend to be ignored by reviewers. For observation 6, we can conclude that patches that modify files whose prior patches had received either few reviewers or short discussions tend not to be discussed, despite being approved by there is at least one reviewer. These findings suggest that the poor review participation in the past (either few reviewers or short discussions) can lead to the poor review participation in the future.

Feedback Delay of Prior Patches Observation 8 shows that the feedback delay of prior patches shares an increasing relationship with the likelihood that a patch that will receive slow initial feedback. Prior work shows that reviews often receive prompt initial feedback. Otherwise, patches tend to be ignored if they have not receive initial feedback for a long period of time (Rigby et al. 2008).

Our observations 1, 6, and 8 have shown that the past review participation shares a link to poor review participation of a patch. Furthermore, our prior study has found that practitioners tend to overlook the history of files, i.e., reviewers did not give much attention to patches made to files that have been historically defective (Thongtanunam et al. 2015a). Therefore, our findings of past review participation metrics suggest that practitioners should take the history of files into consideration in order to break the cycle of poor review participation.

5.2 Past Activity

Past Involvement of Practitioners Observation 7 shows that the past involvement of an author shares an inverse relationship with the likelihood that a patch will be discussed, while the past involvement of reviewers shares an increasing relationship with the likelihood. The relationship of the past involvement of reviewers arrives at our conjecture. However, the relationship of the past involvement of authors is counter our conjecture. A potential reason is that the experience developers are less likely to submit defective patches as those developers have written many patches to those files (Bird et al. 2011).

Rigby et al. (2014) report that the past involvement of author and reviewers have a small effect on the amount of discussion in the email-based code reviews. Yet, their study did not consider the patches that have no review discussion. Hence, this observation can complement the prior study by showing that the past involvement of an author and reviewers shares a link to the likelihood that a patch that will be discussed. Moreover, Kononenko et al. (2015) report that reviewer experience is a good indicator of whether the patch will be effectively reviewed.

Furthermore, the past involvement of reviewers has been recently used to suggest reviewers for a new patch in several studies (Balachandran 2013; Thongtanunam et al. 2015b; Zanjani et al. 2015), and also in commercial MCR tools such as CodeCollaborator (Ratcliffe 2009). Hence, observation 7 can support their approaches that inviting reviewers who have been

involved in many reviews of the modified files can increase the likelihood that a patch will be discussed.

The Number of Days Since the Last Modification Our observation 2 shows that the number of days since the last modification shares an increasing relationship with the likelihood that a patch will have at least one reviewer. This finding does not match our expectations. One possible reason for this relationship is that the last patch prior to the current patch is incomplete. For example, the Android patch of review ID 39850 was approved by a reviewer.[10] Then, two hours later, the patch author submitted a new patch in order to make a change that was suggested in review ID 39850.[11] We find similar examples in the Qt and OpenStack projects.[12] Although such patches can be minor changes or may not need much involvement from reviewers, prior work suggests that the patches still should be examined by at least one reviewers to decrease the likelihood of having defects in the future (Bavota and Russo 2015).

5.3 Patch Properties

Patch Description As we conjecture, observation 3 shows that the description length shares an increasing relationship with the likelihood that a patch will have at least one reviewer. Similarly, observation 5 shows that a short description can lower the likelihood that a patch will be discussed. We find that most of the patches with short descriptions provide neither the details that are necessary to understand the proposed changes nor a link for additional information. For example, the patch author of OpenStack review ID 29856 did not describe the detail of a change.[13] This finding is consistent with prior studies of email-based code reviews, i.e., a descriptive subject and a well-explained change log message are very important information for developers to select patches to review (Rigby and Storey 2011). Moreover, Tao et al. (2012) report that one of the most important pieces of information for reviewers is a description of the rationale of a change.

Observation 9 shows that as we conjecture, a patch that introduces new functionality is more likely to receive slow initial feedback than a patch with another purposes. A potential reason for this delayed feedback in patches that introduce new features could be that such patches require more effort to understand. Intuitively, a documentation patch would be easy to understand, and could receive prompt feedback. We also observe that the reviewers of the bug-fixing patches often are reporters in the Issue Tracking System (ITS). For example, Qt review ID 29856 is made to address the bug ID 22625.[14] We find that the reviewer of this patch is the one who reports the bug.[15] Hence, it is more likely that the bug fix reviewers would already understand the problem and could provide prompt feedback to the author. On the other hand, the purpose of patches that introduce new functionality must be entirely built from the patch description and source code. For example, the patch author

[10] https://android-review.googlesource.com/#/c/39850.

[11] https://android-review.googlesource.com/#/c/39881.

[12] An example in the Qt project: https://codereview.qt-project.org/#/c/27218 and https://codereview.qt-project.org/#/c/30591. An example in the OpenStack project: https://review.openstack.org/#/c/36808 and https://review.openstack.org/#/c/36832.

[13] https://review.openstack.org/#/c/36901/.

[14] https://codereview.qt-project.org/#/c/29856.

[15] https://bugreports.qt.io/browse/QTBUG-22625.

in Qt review ID 101316 implements a new function to `QGeoShape`.[16] Hence, there are likely cases that reviewers will require longer time to understand before providing initial feedback. These observations suggest that to increase review participation, an author should provide a detailed description of their proposed changes in order to help reviewers to understand which problem it fixes or how the new feature is supposed to work. Then, reviewers can either provide feedback if they have the expertise to do so or suggest appropriate reviewers.

Patch Size Observation 4 shows that as we conjecture, churn shares an increasing relationship with the likelihood that a patch will be discussed. Recent studies also report that the patch size is a good indicator of patch acceptance (Weißgerber et al. 2008; Jiang et al. 2013). Intuitively, the large patches are likely to be discussed since they can contain more problems than the small patches. For example, OpenStack review ID 35074 shows a review where an author proposes a change of 737 LOC, then the reviewers raised several issues.[17] On the other hand, smaller patches have less code to critique, and thus are less likely to receive comments from reviewers.[18] This observation is also consistent with the findings of Baysal et al. (2015) who find that large patches tend to have more revisions than small patches in the MCR processes of the WebKit and Blink projects.

6 Threats to Validity

We now discuss threats to the validity of our study.

6.1 External Validity

We focus our study on three open source projects, which may limit the generalizability of our results. Nagappan et al. (2013) argue that increasing the sample size without careful selection cannot contribute to the goal of increased generality. Hence, it is a challenge to carefully identify projects that satisfy our selection criteria (cf. Section 3.1), since the code review process of MCR is a relatively new development. To aid in future work, we make our datasets publicly available.[19] Nonetheless, additional replication studies are needed to generalize our results.

6.2 Construct Validity

We identify the purpose of a patch by extracting keywords from its commit message. Although modern Issue Tracking Systems (ITSs) provide a field for practitioners to denote the purpose of a change, we find that our studied projects have a small proportion of patches that can be linked to records in ITSs. Indeed, only 9 %, 1 %, and 19 % of the studied patches can be linked to issues in the ITS records of our studied projects. Hence, we must rely on

[16] https://codereview.qt-project.org/#/c/101316.

[17] https://review.openstack.org/#/c/35074/.

[18] https://review.openstack.org/#/c/36448/.

[19] http://sailhome.cs.queensu.ca/replication/review_participation/

heuristics to recover this information. Nevertheless, we measure the accuracy of our purpose identification by manually examining samples of patches. From a sample of 50 patches for each type of purpose, we find that on average, 89 % of patches are correctly identified as feature introduction, 91 % of patches are correctly identified as bug-fixing, and 75 % of patches are correctly identified as documentation. We provide online access to our samples and their manual classification results.[20]

We measure feedback delay based on a heuristic that reviewers will promptly review a new patch at the time that it is submitted. However, there are likely cases where reviewers actually examined a patch for a fraction of this timeframe. Unfortunately, reviewers do not record the time that they actually spent reviewing a patch. To reduce such measurement errors, we use the elapsed time between the latest revision before receiving the initial feedback and the posting of the initial feedback.

6.3 Internal Validity

We assume that our studied projects perform code reviews using the MCR tools. There are likely cases where the reviewing activities are missing. For example, Mukadam et al. (2013) report that many reviews in the Android project are missing since August 2011 until early January 2012. It is also possible that reviewers may provide feedback using other communication media, such as in-person discussion (Beller et al. 2014), a group IRC (Shihab et al. 2009), or mailing list (Guzzi et al. 2013; Rigby and Storey 2011). Unfortunately, recovering these reviewing activities is a non-trivial problem (Bacchelli et al. 2010; Bird et al. 2007). However, our analysis focuses on reviews that were submitted during the period when the studied projects actively use MCR tools (cf. Section 3.2.1). Hence, we rely on the information that is recorded by these tools.

Since our observations are based on the surviving explanatory variables, there are likely cases that our variable selection may influence our conclusions. Hence, for the sake of completeness, we change each surviving variable to the other variables that were removed and refit our models. Our model analysis results indicate that these alternate models achieve similar AUC values, i.e., the AUC differences are ranging between -0.01 to 0.04 for models in RQ1, -0.01 to 0.02 for models in RQ2, and -0.01 to 0.01 for models in RQ3. Furthermore, we find that if the surviving variables have a large effect on the likelihood, the alternate variables will have a large effect as well. For example, Table 6 shows that the number of reviewers of prior patches has the largest negative effect on the likelihood that a patch will not attract reviewer. We also find that the discussion length of prior patches has the largest negative effect on the likelihood in the alternate model (i.e., the variable that is highly correlated with the number of reviewers of prior patches). Moreover, the variables that are highly correlated in our study measure similar characteristics of a patch, e.g., the number of reviewers of prior patches and the discussion length of prior patches are measures of past review participation. Therefore, we believe that our variable selection does not muddle our conclusions.

We assume that the review processes are consistent across all subsystems in a large project. Future work should closely examine whether there are differences in review processes across subsystems.

[20] http://sailhome.cs.queensu.ca/replication/review_participation/.

7 Conclusion

Due to the human-intensive nature of code reviewing, review participation plays an important role in Modern Code Review (MCR) practices. Despite the importance of review participation (McIntosh et al. 2014; Thongtanunam et al. 2015a; Morales et al. 2015), little is known about the factors that influence review participation in the MCR process.

In this paper, we investigate the characteristics of patches that: do not attract reviewers, are not discussed, or receive slow initial feedback in the MCR process. We measure 20 patch and MCR process metrics grouped along five dimensions. We use contemporary regression modelling techniques to investigate the relationship that our metrics share with the likelihood that a patch will suffer from poor review participation. Using data collected from the Android, Qt, and OpenStack open source projects, we empirically study 196,712 code reviews. The results of our study show that our models can identify patches that fill suffer from poor review participation with an AUC ranging 0.61-0.76. Moreover, we make the following observations:

- The number of reviewers of prior patches, and the description length share a strong inverse relationship with the likelihood that a patch will not attract any reviewers. Counter-intuitively, the number of days since the last modification of files also share an inverse relationship. (Observations 1-3).
- The description length, the discussion length of prior patches, churn, and past involvement of reviewers share an inverse relationship with the likelihood that a patch will not be discussed, while past involvement of an author shares an increasing relationship with that likelihood (Observations 4-7).
- The feedback delay of prior patches shares a strong increasing relationship with the likelihood that a patch will receive slow initial feedback. Moreover, patches that introduce new features tend to receive slower feedback than patches that fix bugs or address documentation issues (Observations 8-9).

We believe that our results and empirical observations help to support the management of the MCR process and adherence to our recommendations will lead to a more responsive review process. To facilitate future work, we provide online access to our patch data and example R scripts for model our construction and analysis approaches.[21]

Acknowledgments This research was supported by the Grant-in-Aid for JSPS Fellows (Numbers 16J02861).

References

Abelein U, Paech B (2013) Understanding the influence of user participation and involvement on system success a systematic mapping study. Empir Softw Eng (EMSE) 20(1):28–31

Bacchelli A, Bird C (2013) Expectations, Outcomes, and Challenges of Modern Code Review. In: Proceedings of the 35th International Conference on Software Engineering (ICSE), pp 712–721

Bacchelli A, Lanza M, Robbes R (2010) Linking E-Mails and Source Code Artifacts. In: Proceedings of the 32nd International Conference on Software Engineering (ICSE), pp 375–384

[21] http://sailhome.cs.queensu.ca/replication/review_participation/.

Balachandran V (2013) Reducing Human Effort and Improving Quality in Peer Code Reviews using Automatic Static Analysis and Reviewer Recommendation. In: Proceedings of the 35th International Conference on Software Engineering (ICSE), pp 931–940
Bavota G, Russo B (2015) Four Eyes Are Better Than Two: On the Impact of Code Reviews on Software Quality. In: Proceedings of the 31st International Conference on Software Maintenance and Evolution (ICSME), pp 81–90
Baysal O, Kononenko O, Holmes R, Godfrey MW (2012) The Secret Life of Patches: A Firefox Case Study. In: Proceedings of the 19th Working Conference on Reverse Engineering (WCRE), pp 447–455
Baysal O, Kononenko O, Holmes R, Godfrey MW (2015) Investigating technical and non-technical factors influencing modern code review. Empirical Software Engineering (EMSE) 1–28
Beller M, Bacchelli A, Zaidman A, Juergens E (2014) Modern Code Reviews in Open-Source Projects: Which Problems Do They Fix? In: Proceedings of the 11th Working Conference on Mining Software Repositorie (MSR), pp 202–211
Bettenburg N, Hassan AE, Adams B, German DM (2013) Management of community contributions - A case study on the Android and Linux software ecosystems. Empirical Software Engineering (EMSE) 1–38
Bird C, Gourley A, Devanbu P, Gertz M, Swaminathan A (2006) Mining Email Social Networks. In: Proceedings of the 3rd International Workshop on Mining Software Repositories (MSR), pp 137–143
Bird C, Gourley A, Devanbu P (2007) Detecting Patch Submission and Acceptance in OSS Projects. In: Proceedings of the 4th International Workshop on Mining Software Repositories (MSR), pp 26–29
Bird C, Nagappan N, Murphy B, Gall H, Devanbu P (2011) Don't Touch My Code! Examining the Effects of Ownership on Software Quality. In: Proceedings of the 8th joint meeting of the European Software Engineering Conference and the Internaltional Symposium on the Foundations of Software Engineering (ESEC/FSE), pp 4–14
Bosu A, Carver JC (2013) Impact of Peer Code Review on Peer Impression Formation: A Survey. In: Proceedings of the 7th International Symposium on Empirical Software Engineering and Measurement (ESEM), pp 133–142
Cramėr H (1999) Mathematical methods of statistics. University Press, Princeton, p 9
Efron B (1986) How biased is the apparent error rate of a prediction Rule? J Am Stat Assoc 81(394):461–470
Fagan ME (1999) Design and code inspections to reduce errors in program development. IBM Syst J 38(2-3):258–287
Fowler M, Foemmel M (2006) Continuous integration. http://wwwthoughtworkscom/ContinuousIntegrationpdf
Gousios G, Pinzger M, van Deursen A (2014) An Exploratory Study of the Pull-based Software Development Model. In: Proceedings of the 36th International Conference on Software Engineering (ICSE), pp 345–355
Guzzi A, Bacchelli A, Lanza M, Pinzger M, Van Deursen A (2013) Communication in Open Source Software Development Mailing Lists. In: Proceedings of the 10th International Working Conference on Mining Software Repositories (MSR), pp 277–286
Hamasaki K, Kula RG, Yoshida N, Erika CCA, Fujiwara K, Iida H (2013) Who does what during a Code Review? An extraction of an OSS Peer Review Repository. In: Proceedings of the 10th International Working Conference on Mining Software Repositories (MSR), pp 49–52
Hanley JA, McNeil BJ (1982) The meaning and use of the area under a receiver operating characteristic (ROC) curve. Radiological Society of North America 143(1):29–36
Harrell Jr FE (2002) Regression Modeling Strategies: With Application to Linear Models, Logistic Regression, and Survival Analysis, 1st. Springer, Berlin
Harrell Jr FE (2015) rms: Regression Modeling Strategies. http://biostat.mc.vanderbilt.edu/rms
Hassan AE (2008) Automated Classification of Change Messages in Open Source Projects. In: Proceedings of the 23rd Symposium on Applied Computing (SAC), pp 837–841
Hassan AE (2009) Predicting Faults Using the Complexity of Code Changes. In: Proceedings of the 31st International Conference on Software Engineering (ICSE), pp 78–88
Hinkle DE, Wiersma W, Jurs SG (1998) Applied statistics for the behavioral sciences, 4th edn. Houghton Mifflin, Boston
Jiang Y, Adams B, German DM (2013) Will My Patch Make It? And How Fast? Case Study on the Linux Kernel. In: Proceeding of the 10th International Working Conference on Mining Software Repositories (MSR), pp 101–110
Kamei Y, Shihab E, Adams B, Hassan AE, Mockus A, Sinha A, Ubayashi N (2013) A large-scale empirical study of just-in-time quality assurance. Trans Softw Eng (TSE) 39(6):757–773
Kim S, Whitehead EJJr, Zhang Y (2008) Classifying software changes: Clean or buggy? Trans Softw Eng (TSE) 34(2):181–196

Kononenko O, Baysal O, Guerrouj L, Cao Y, Godfrey MW (2015) Investigating Code Review Quality : Do People and Participation Matter ? In: Proceedings of the 31st International Conference on Software Maintenance and Evolution (ICSME), pp 111–120

McIntosh S, Kamei Y, Adams B, Hassan AE (2014) The Impact of Code Review Coverage and Code Review Participation on Software Quality. In: Proceedings of the 11th International Working Conference on Mining Software Repositorie (MSR), pp 192–201

McIntosh S, Kamei Y, Adams B, Hassan AE (2015) An Empirical Study of the Impact of Modern Code Review Practices on Software Quality. Empirical Software Engineering (EMSE)

Mishra R, Sureka A (2014) Mining Peer Code Review System for Computing Effort and Contribution Metrics for Patch Reviewers. In: Proceedings of the 4th Workshop on Mining Unstructured Data (MUD), pp 11–15

Mockus A, Votta LG (2000) Identifying Reasons for Software Changes using Historic Databases. In: Proceedings of the 16th International Conference on Software Maintainance (ICSM), pp 120–130

Morales R, Mcintosh S, Khomh F (2015) Do Code Review Practices Impact Design Quality? A Case Study of the Qt, VTK, and ITK Projects. In: Proceedings of the 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)

Mukadam M, Bird C, Rigby PC (2013) Gerrit software code review data from android. In: Proceedings of the 10th International Working Conference on Mining Software Repositories (MSR), pp 45–48

Nagappan M, Zimmermann T, Bird C (2013) Diversity in Software Engineering Research. In: Proceedings of the 9th joint meeting of the European Software Engineering Conference and the International Symposium on the Foundations of Software Engineering (ESEC/FSE), pp 466–476

Nagappan N, Zeller A, Zimmermann T, Herzig K, Murphy B (2010) Change bursts as defect predictors. In: Proceedings of the 21st International Symposium on Software Reliability Engineering (ISSRE), pp 309–318

Nurolahzade M, Nasehi SM, Khandkar SH, Rawal S (2009) The Role of Patch Review in Software Evolution: An Analysis of the Mozilla Firefox. In: Proceedings of the join International and Anual ERCIM Workshops on Principles of Software Evolution and Software Evolution Workshop (IWPSE-Evol), pp 9–17

Porter A, Siy H, Mockus A, Votta L (1998) Understanding the sources of variation in software inspections. Trans Softw Eng Methodol (TOSEM) 7(1):41–79

Ratcliffe JG (2009) Moving Software Quality Upstream: The Positive Impact of Lightweight Peer Code Review. In: Pacific NW Software Quality Conference, pp 1–10

Raymond ES (1999) The cathedral and the bazaar. Knowl, Technol Policy 12(3):23–49

Rea LM, Parker RA (2014) Designing and conducting survey research: A comprehensive guide. John Wiley & Sons

Rigby PC, Bird C (2013) Convergent Contemporary Software Peer Review Practices. In: Proceedings of the 9th joint meeting of the European Software Engineering Conference and the International Symposium on the Foundations of Software Engineering (ESEC/FSE), pp 202–212

Rigby PC, Storey MA (2011) Understanding Broadcast Based Peer Review on Open Source Software Projects. In: Proceeding of the 33rd International Conference on Software Engineering (ICSE), pp 541–550

Rigby PC, German DM, Storey MA (2008) Open Source Software Peer Review Practices: A Case Study of the Apache Server. In: Proceedings of the 30th International Conference on Software Engineering (ICSE), pp 541–550

Rigby PC, Cleary B, Painchaud F, Storey MA, German DM (2012) Contemporary peer review in action: Lessons from open source development. IEEE Softw 29(6):56–61

Rigby PC, German DM, Cowen L, Storey MA (2014) Peer review on open-source software projects: Parameters, Statistical Models, and Theory. Transactions on Software Engineering and Methodology (TOSEM) 23(4):Article No. 35

Sarle WS (1990) The VARCLUS procedure, 4th edn. SAS Institute, Inc

Sauer C, Jeffery DR, Land L, Yetton P (2000) The effectiveness of software development technical reviews: a behaviorally motivated program of research. Trans Softw Eng (TSE) 26(1):1–14

Shihab E, Jiang ZM, Hassan AE (2009) Studying the Use of Developer IRC Meetings in Open Source Projects. In: Prceedings of the 25th International Conference on Software Maintenance (ICSM), pp 147–156

Shull F, Basili V, Boehm B, Brown AW, Costa P, Lindvall M, Port D, Rus I, Tesoriero R, Zelkowitz M (2002) What We Have Learned About Fighting Defects. In: Proceedings of the 8th International Software Metrics Symposium (METRICS), pp 249–258

Tantithamthavorn C, McIntosh S, Hassan AE, Ihara A, Matsumoto K (2015) The Impact of Mislabelling on the Performance and Interpretation of Defect Prediction Models. In: Proceedings of the 37th International Conference on Software Engineering (ICSE), pp 812–823

Tao Y, Dang Y, Xie T, Zhang D, Kim S (2012) How Do Software Engineers Understand Code Changes?: An Exploratory Study in Industry. In: Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering (FSE), pp 51:1–51:11

Thongtanunam P, Yang X, Yoshida N, Kula RG, Ana Erika CC, Fujiwara K, Iida H (2014) ReDA: A Web-based Visualization Tool for Analyzing Modern Code Review Dataset. In: The proceeding of the 30th International Conference on Software Maintenance and Evolution (ICSME), pp 606–609

Thongtanunam P, McIntosh S, Hassan AE, Iida H (2015a) Investigating Code Review Practices in Defective Files: An Empirical Study of the Qt System. In: Proceedings of the 12th International Working Conference on Mining Software Repositories (MSR), pp 168–179

Thongtanunam P, Tantithamthavorn C, Kula RG, Yoshida N, Iida H, Matsumoto K (2015b) Who Should Review My Code? A File Location-Based Code-Reviewer Recommendation Approach for Modern Code Review. In: Proceedings of the 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), pp 141–150

Thongtanunam P, McIntosh S, Hassan AE, Iida H (2016) Revisiting Code Ownership and its Relationship with Software Quality in the Scope of Modern Code Review. In: Proceedings of the 38th International Conference on Software Engineering (ICSE), pp 1039–1050

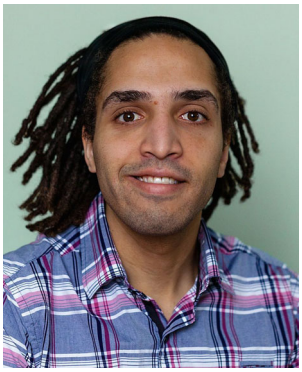
Tsay J, Dabbish L, Herbsleb J (2014) Let's Talk About It: Evaluating Contributions through Discussion in GitHub. In: Proceedings of the 22nd International Symposium on the Foundations of Software Engineering (FSE), pp 144–154

Ukkonen E (1985) Algorithms for approximate string matching. Inf Control 64(1-3):100–118

Weißgerber P, Neu D, Diehl S (2008) Small Patches Get In ! In: Proceedings of the 2008 international working conference on Mining software repositories (MSR'08), pp 67–75

Zanjani M, Kagdi H, Bird C (2015) Automatically recommending peer reviewers in modern code review. IEEE Trans Softw Eng:1–13

Patanamon Thongtanunam is a research fellow (DC2) of the Japan Society for the Promotion of Science (JSPS) and Ph.D. candidate at Nara Institute of Science and Technology (NAIST), Japan. She received the B.E. degree (2012) in computer engineering from Kasetsart University, Thailand, and the M.E. degree (2014) in Information Science from Nara Institute of Science and Technology, Japan. Her research is focused on uncovering empirical evidence and knowledge from data recorded in software repositories by using statistical analysis. Her Ph.D. thesis aims to better understand the current practices of code review and to provide actionable insights and process support for practitioners. In addition to the work presented here, the foundation for her Ph.D. thesis has been published at International Conference on Software Engineering (ICSE), International Conference on Mining Software Repositories (MSR), and International Conference on Software Analysis, Evolution, and Reengineering (SANER). More about Patanamon and her work is available online at http://patanamon.com/.



Shane McIntosh is an assistant professor in the Department of Electrical and Computer Engineering at McGill University. He received his Bachelor's degree in Applied Computing from the University of Guelph and his MSc and Ph.D. in Computer Science from Queen's University. In his research, Shane uses empirical software engineering techniques to study software build systems, release engineering, and software quality. His research has been published at several top-tier software engineering venues, such as the International Conference on Software Engineering (ICSE), the International Symposium on the Foundations of Software Engineering (FSE), and the Springer Journal of Empirical Software Engineering (EMSE). More about Shane and his work is available online at http://shanemcintosh.org/.

Ahmed E. Hassan is the Canada Research Chair (CRC) in Software Analytics, and the NSERC/BlackBerry Software Engineering Chair at the School of Computing at Queen's University, Canada. His research interests include mining software repositories, empirical software engineering, load testing, and log mining. Hassan received a Ph.D. in Computer Science from the University of Waterloo. He spearheaded the creation of the Mining Software Repositories (MSR) conference and its research community. Hassan also serves on the editorial boards of IEEE Transactions on Software Engineering, Springer Journal of Empirical Software Engineering, Springer Journal of Computing, and PeerJ Computer Science. Contact ahmed@cs.queensu.ca. More information at: http://sail.cs.queensu.ca/.

Hajimu Iida received his B.E., M.E., and Dr. of Eng. degrees from Osaka University in 1988, 1990, and 1993, respectively. From 1991 to 1995, he worked for the Department of Information and Computer Science, Faculty of Engineering Science, Osaka University as a research associate. Since 1995 he has been with the Graduate School of Information Science, Nara Institute of Science and Technology, Japan. His current position is a Professor of the Laboratory of Software Design and Analysis. His research interests include modeling and analysis of software and development process.