Empirical Software Engineering (2021) 26: 96
https://doi.org/10.1007/s10664-021-09997-x

Understanding shared links and their intentions to meet information needs in modern code review:

A case study of the OpenStack and Qt projects

Dong Wang[1] · Tao Xiao[1] · Patanamon Thongtanunam[2] · Raula Gaikovina Kula[1] · Kenichi Matsumoto[1]

Accepted: 1 June 2021 / Published online: 8 July 2021

Abstract
Code reviews serve as a quality assurance activity for software teams. Especially for Modern Code Review, sharing a link during a review discussion serves as an effective awareness mechanism where "Code reviews are good FYIs [for your information].". Although prior work has explored link sharing and the information needs of a code review, the extent to which links are used to properly conduct a review is unknown. In this study, we performed a mixed-method approach to investigate the practice of link sharing and their intentions. First, through a quantitative study of the OpenStack and Qt projects, we identify 19,268 reviews that have 39,686 links to explore the extent to which the links are shared, and analyze a correlation between link sharing and review time. Then in a qualitative study, we manually analyze 1,378 links to understand the role and usefulness of link sharing. Results indicate that internal links are more widely referred to (93% and 80% for the two projects). Importantly, although the majority of the internal links are referencing to reviews, bug reports and source code are also shared in review discussions. The statistical models show that the number of internal links as an explanatory factor does have an increasing relationship with the review time. Finally, we present seven intentions of link sharing, with providing context being the most common intention for sharing links. Based on the findings and a developer survey, we encourage the patch author to provide clear context and explore both internal and external resources, while the review team should continue link sharing activities. Future research directions include the investigation of causality between sharing links and the review process, as well as the potential for tool support.

Keywords Code review · Mining software repositories · Link sharing · Information needs

Communicated by: Massimiliano Di Penta

✉ Dong Wang
wang.dong.vt8@is.naist.jp

Extended author information available on the last page of the article.

1 Introduction

Software code reviews serve as quality assurance for software teams (Huizinga and Kolawa 2007; Rigby and Storey 2011). From being a formal code inspection process conducted by face-to-face meetings (Fagan 1976), nowadays the Modern Code Review (MCR) process becomes more flexible with asynchronous collaboration through an online tool (such as Gerrit,[1] Codestriker,[2] and ReviewBoard[3]). These online tools are now widely adopted in both open source and proprietary software projects (Rigby and Bird 2013; Sadowski et al. 2018). Not only improving the overall quality of a patch (i.e., software changes), Bacchelli and Bird (2013) also reported that MCR also serves as an effective mechanism to increase awareness and share information: "Code reviews are good FYIs [for your information].".

An effective review requires proper understanding. However, it is challenging to identify and acquire the needed information to have a proper understanding to conduct a review. This is especially a case for a large software project like OpenStack, which has code reviews of over 20 million lines of codes that are submitted by over 100,000 contributors spread more than 600 code repositories (Zhang et al. 2019). Pascarella et al. (2018) find that during reviews, reviewers often request additional information about correct understanding, alternative solution, to improve patch quality. Ebert et al. (2019) report that reviewers often suffer from confusion due to a lack of information about the intention of a patch (i.e., a rationale for a change).

Recent work points out that the link shared in the review discussion can be used to provide information related to a review. Hirao et al. (2019) show that shared links between reviews can be used to indicate the information about patch dependency, broader context, and alternative solution. However, the other types of links other than review links were not studied in the work of Hirao et al. (2019). On the other hand, Jiang et al. (2019) conducted a quantitative analysis to find developers that share various types of links in ten GitHub projects. However, this study does not systematically investigate the correlation between the shared links and the review process, and lacks of qualitative analysis of why these various types of links are shared.

Based on the findings of prior work (Hirao et al. 2019), we hypothesize that sharing links may help developers fulfill the information needs. Yet, it is still unclear about what types of information could be fulfilled by these various types of links. To fill this gap, this work aims to explore the prevalence of link sharing, systematically investigate the correlation between link sharing and review time, and qualitatively investigate what are the intentions of sharing links. In this paper, the intention is defined as the intention of sharing a link to meet a certain type of information needed during code review. Through a case study of the OpenStack and Qt projects (two large-scale and thriving open source projects with globally distributed teams that actively perform code reviews), we identify 19,268 reviews that have 39,686 links shared during review discussions. We formulate three research questions to guide our study:

– **RQ1: To what extent do developers share links in the review discussion?** It is not yet known how a project of repositories uses link sharing in their communities. Using a sample of well-known OpenStack and Qt projects, we would like to investigate the trend of link sharing, common domains in the project, and the link target types.

[1] https://www.gerritcodereview.com/
[2] http://codestriker.sourceforge.net/
[3] https://www.reviewboard.org/

- **RQ2: Does the number of links shared in the review discussion correlate with review time?** Prior studies (Baysal et al. 2016; Kononenko et al. 2018) analyzed the impact of technical and non-technical factors on the review process (e.g., review outcome, review time). However, little is known about whether or not the practice of sharing links can be correlated with review time. It is possible that link sharing may shorten the review time as it provides the required information to a review, which might help reviewers to conduct a review faster. To address this RQ, we conduct a statistical analysis using a non-linear regression model to analyze a correlation between link sharing and review time.
- **RQ3: What are the common intentions of links shared in the review discussion?** Previous work (Pascarella et al. 2018) has identified different types of information that are needed by reviewers when conducting a review. Yet, little is known to what extent can link sharing meets such information needs. Hence, we aim to investigate the intention behind link sharing in order to better understand the role and usefulness of link sharing during reviews.

The key results of each RQ are as follow: *For RQ1*, our results show that in the past five years, 25% and 20% of the reviews have at least one link shared in a review discussion within the OpenStack and Qt. 93% and 80% of shared links are the internal links that are directly related to the project. Importantly, although the majority of the internal links are referencing to reviews, bug reports, and source code are also shared in review discussions. *For RQ2*, our non-linear regression model results show that the internal link has a significant correlation with the code review time. However, the external link is not significantly correlated, for OpenStack and Qt. Furthermore, we observe that the number of internal links has an increasing relationship with the review time. *For RQ3*, we identify seven intentions of sharing links: (1) Providing Context, (2) Elaborating, (3) Clarifying, (4) Explaining Necessity, (5) Proposing Improvement, (6) Suggesting Experts, and (7) Informing Splitted Request. Specifically, for the internal links, we observe that the most popular intention is to provide context. While for the external links, to elaborate the review discussions (i.e., provide a reference or related information) is the most common intention.

The results lead us to conclude that link sharing is increasingly used as a mechanism of sharing information in a code review process, the number of internal links has a positive correlation with the review time, and the intention of link sharing is often used to provide context understanding. *For patch authors*, they should provide a clear context of the patch by sharing links (i.e., containing implementation related information) for reviewer teams to better understand a patch. *For review teams*, link sharing should be encouraged, as results indicate that it can fulfill information needs and contains crucial knowledge to assist the author, which will support a more efficient review process. *For researchers*, with the increasing usage of links, there is an opportunity for tool support for suggesting shared links to retrieve useful information for both patch authors and review teams. The study contributions are three-fold: (i) a large quantitative and qualitative study on link sharing on the MCR process, (ii) a taxonomy of seven intentions of sharing links, and (iii) a full replication of our study, including the scripts and datasets. [4]

The remainder of this paper is organized as follows: Section 2 introduces the background of the study. Section 3 describes the studied projects, the data preparation, and the analysis approach for each research question. Section 4 reports the results of our empirical study.

[4]https://github.com/NAIST-SE/LinkIntentioninCR/

Section 5 discusses the implications from our findings. Section 6 discloses the threats to validity and Section 7 presents the related work. Finally, we conclude the paper in Section 8.

2 Motivating Example

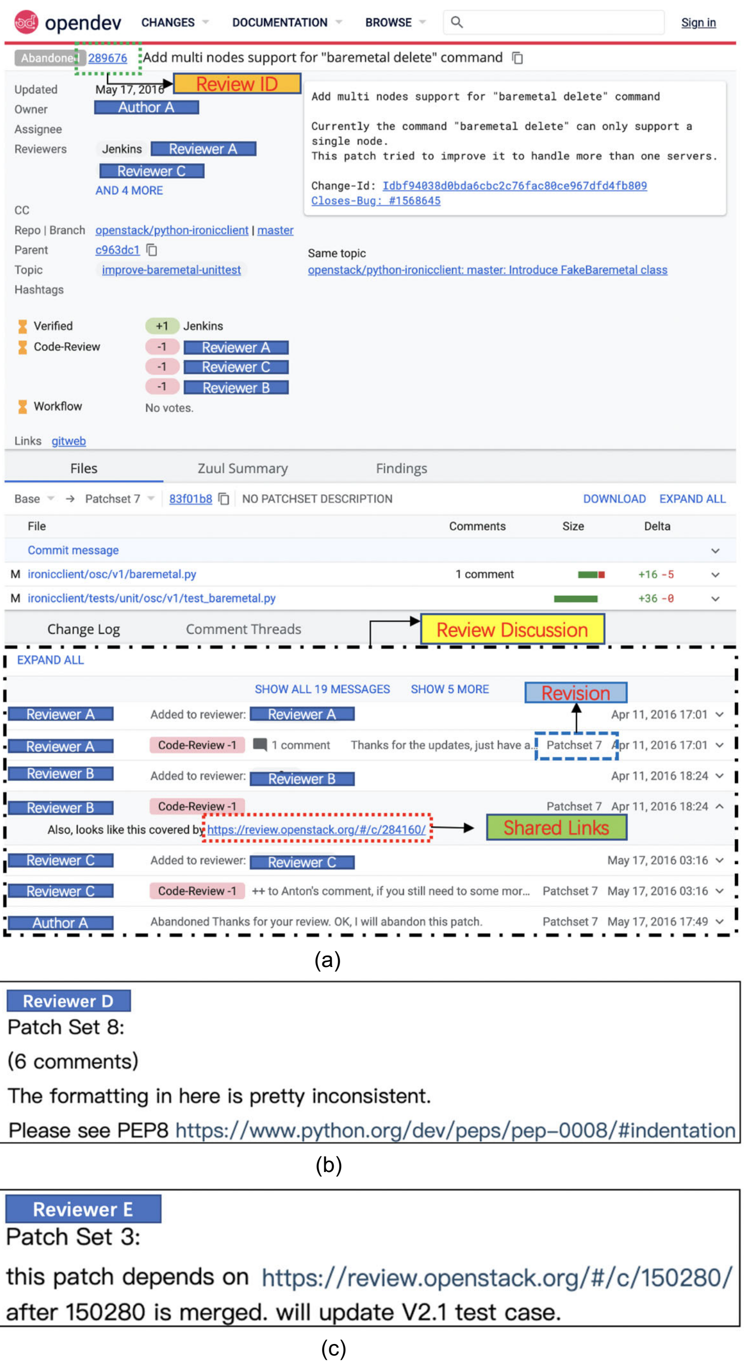
We highlight two challenges in identifying information needs in Code Review. The first challenge is that the rationale to meet information needs is unclear. Prior studies found that missing a rationale and a lack of familiarity with existing code (e.g., a lack of knowledge about the code that's being modified) are the most prevalent reasons for causing discussion confusion and low reviewer participation in code reviews (Ebert et al. 2019; Ruangwan et al. 2018). Such confusion can delay the incorporation of a code change into a code base. Patch description is another vital information to help developers understand the changes. Tao et al. (2012) stated that one of the most important pieces of information for reviewers is a description of the rationale of a change. In addition, the description length shares an increasing relationship with the likelihood of reviewer participation (Thongtanunam et al. 2017). The second challenge is understanding which information is needed to facilitate the review. As reported by Bacchelli and Bird (2013), understanding is the main challenge for developers when doing code reviews. The most difficult task from the understanding perspective is finding defects, immediately followed by alternative solutions. They point out that context and change understanding must be improved if managers and developers want to match their needs. Recently, Pascarella et al. (2018) highlights the presence of seven types of high-level information needs, with the two most prominent being the needs to know (1) whether a proposed alternative solution is valid and (2) whether the understanding of the reviewer about the code under review is correct.

To help developers find related changes, the code review tool like Gerrit has provided functionalities that allow a patch author to share the links of reviews that have related changes or in the same topics.[5] Yet, we observe that developers still share links in the review discussion. Figure 1 shows three motivating examples of how sharing links in the review discussion can be a means to fill in the needed information. The first observation is that various links can be shared in a review discussion to provide information. For instance, as shown in Fig. 1a, the reviewer *Anton Arefiev* shared a link of a review in the PatchSet 7 (i.e., the seventh revision of the review #289676) to inform the patch author of the review #289676 that the review #284160 covered the current review #289676. Figure 1b shows another example where the reviewer *melanle witt* shared a link of Python documentation in a review discussion of the review #207794 in order to improve the coding format.

While the work of Hirao et al. (2019) extensively investigate the review links shared in code reviews, these motivating examples show that links that point to other information sources are also shared in code reviews. Hence, in this study, we aim to investigate the trend and characteristics of shared links in terms of their types (internal or external) and the kinds of content to which those links point. In this study, we define 'internal links' are the links that are directly related to the project (e.g., review links, bug reports, git repository), while external links are the links that refer to resources outside the project (e.g., Python Document).

Apart from the various links, the second observation is that the intentions behind sharing links can be different even if the shared links have the same type. In Fig. 1a, the intention

[5]https://gerrit-review.googlesource.com/Documentation/user-review-ui.html



Abandoned 289676 Add multi nodes support for "baremetal delete" command

Updated May 17, 2016
Owner Author A
Assignee
Reviewers Jenkins Reviewer A Reviewer C AND 4 MORE
CC
Repo | Branch openstack/python-ironicclient | master
Parent c963dc1
Topic improve-baremetal-unittest
Hashtags

```
Add multi nodes support for "baremetal delete" command

Currently the command "baremetal delete" can only support a
single node.
This patch tried to improve it to handle more than one servers.

Change-Id: Idbf94038d0bda6cbc2c76fac80ce967dfd4fb809
Closes-Bug: #1568645
```

Same topic
openstack/python-ironicclient: master: Introduce FakeBaremetal class

Verified	+1	Jenkins
Code-Review	-1	Reviewer A
	-1	Reviewer C
	-1	Reviewer B
Workflow	No votes.	

Links gitweb

Base → Patchset 7 83f01b8 NO PATCHSET DESCRIPTION

	File	Comments	Size	Delta
	Commit message			
M	ironicclient/osc/v1/baremetal.py	1 comment		+16 -5
M	ironicclient/tests/unit/osc/v1/test_baremetal.py			+36 -0

EXPAND ALL
SHOW ALL 19 MESSAGES SHOW 5 MORE

Reviewer A	Added to reviewer: Reviewer A		Apr 11, 2016 17:01
Reviewer A	Code-Review -1 1 comment Thanks for the updates, just have a...	Patchset 7	Apr 11, 2016 17:01
Reviewer B	Added to reviewer: Reviewer B		Apr 11, 2016 18:24
Reviewer B	Code-Review -1 Also, looks like this covered by https://review.openstack.org/#/c/284160/	Patchset 7	Apr 11, 2016 18:24
Reviewer C	Added to reviewer: Reviewer C		May 17, 2016 03:16
Reviewer C	Code-Review -1 ++ to Anton's comment, if you still need to some mor...	Patchset 7	May 17, 2016 03:16
Author A	Abandoned Thanks for your review. OK, I will abandon this patch.	Patchset 7	May 17, 2016 17:49

Reviewer D
Patch Set 8:
(6 comments)
The formatting in here is pretty inconsistent.
Please see PEP8 https://www.python.org/dev/peps/pep-0008/#indentation

Reviewer E
Patch Set 3:
this patch depends on https://review.openstack.org/#/c/150280/
after 150280 is merged. will update V2.1 test case.

Fig. 1 Motivating examples of link sharing in MCR process

of the reviewer of sharing a review link is to point out that the current review #289696 is no longer necessary. While, in Fig. 1c, although the reviewer also shared a link to the review #150718, the intention of the reviewer is to help the patch authors clearly understand the context and code dependency of the current patch.

While the work of Jiang et al. (2019) investigate the different types of links shared in pull-based review, our motivating examples show that the intention of providing information can be different even though the types of shared links are the same. Thus, we aim to better understand the intentions behind link sharing and to what extent can link sharing meet the information needs of review teams.

3 Case Study Design

In this section, we describe the studied projects and data preparation. Then we present the analysis approach for each research question.

3.1 Studied Projects

Since we want to study the practice of link sharing, we focus on the projects that use a code review tool. In this study, we select the projects that use the well-known **Gerrit** platform, a review tool that is largely adopted by many open source projects, where the review data is accessible through REST API. From the range of open source projects that are listed in the work of Thongtanunam and Hassan (2020), we start with four projects: OpenStack, Qt, LibreOffice, and Chromium, as these four projects actively perform code reviews through Gerrit. However, we observe that a large proportion of LibreOffice reviews have only one reviewer. For the Chromium project, we find that it is not trivial to analyze the shared links based on their domains (i.e., most are under the google.com domain), since we want to investigate whether the link is external or not. To gain more insights and avoid potential errors, we exclude LibreOffice and Chromium from our study. Therefore, in this paper, we perform a case study on OpenStack and Qt projects.

OpenStack is an open source software project where many well-known organizations and companies, e.g., IBM, VMware, and NEC, collaboratively develop a platform for cloud computing. Qt project is developed for creating graphical user interfaces as well as cross-platform applications that run on various software and hardware platforms, such as Linux, and Windows.

3.2 Data Preparation

Figure 2 describes an overview of our data preparation. Our data preparation process (DP) consists of two steps: (DP1) clean dataset and (DP2) extract links.

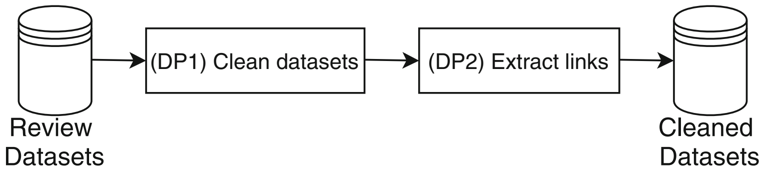


Fig. 2 An overview of data preparation

Table 1 Studied projects

	OpenStack	Qt
Studied Period	11/2011-07/2019	05/2011-07/2019
# Reviews (#Merged/#Abandoned)	58,212 (45,439/12,773)	40,758 (35,284/5,474)
# Reviewers	4,568	1,123
# Reviews with Links	14,655 (25.2%)	4,613 (11.3%)
# Unique Links	26,746	7,518
# Total Links	31,698	7,988
# Links per Review (1st Qu./Median/3rd Qu.)	1/1/2	1/1/2
Percent of Links Shared by Reviewers	62.3%	44.0%
Percent of Links Shared by Authors	37.6%	56.0%

(DP1) Clean dataset For two studied projects, we use the review datasets from the work of Thongtanunam and Hassan (2020). In order to study the correlation between review process duration and the link, we only include the reviews whose status are abandoned or merged. We exclude reviews with open status, since we can not calculate the review time of these reviews. Since we want to investigate the trend of links that are shared in review discussions, we exclude the comments that are posted by automated tools in the discussion thread. We refer to the documentation of the studied systems,[6] to identify the automated tools that are integrated with the code review tools. In addition, we exclude the reviews that do not have comments posted by the reviewers (or have only comments posted by the patch author). Table 1 shows the number of remaining reviews that are studied in this work. For OpenStack, 58,212 reviews are captured from November 2011 to July 2019. Qt owns 40,758 reviews from May 2011 to July 2019.

(DP2) Extract links To identify the links in the review discussion, we apply regular expression (i.e., 'https?://\S+') to search for hyper links in review discussions. Finally, as shown in Table 1, we are able to collect 14,655 reviews with 31,698 links and 4,613 reviews including 7,988 links for OpenStack and Qt projects, respectively. Table 1 provides summary statistics of links per review and whether the links are shared by a patch author or a reviewer. More specifically, in our datasets, 37.6% of links in OpenStack are shared by the patch authors, while 62.3% of links are shared by the reviewers. For Qt, 44% of links are shared by the patch authors, while 56% of links are shared by the patch authors.

3.3 RQ1 Analysis

To answer RQ1: To what extent do developers share links in the review discussion?, we analyze to which extent of links are shared in three main aspects: (1) the trend of link sharing, (2) the common domains, and (3) the types of link targets. Below, we describe our analysis approach for each aspect.

Link Sharing Trend To investigate the trend of link sharing, we examine how often reviews will have link sharing overtime. Similar to prior work (Hirao et al. 2019), we measure the

[6] https://docs.openstack.org/infra/manual/developers.html and https://wiki.qt.io/CIOverview

proportion of reviews that have at least one link shared in the review discussion in an interval of three months.

Common Domain in the Project To analyze the domain popularity, we first determine whether the links are internal (i.e., the links that are directly related to the project), or external (i.e., the links that refer to the resources outside the projects). To identify whether the links are internal or external, we manually examine the domain name and its homepage to determine the links are directly related to the projects (i.e., internal links). More specifically, for OpenStack, we determine the links with the domain names that have the 'openstack', 'opendev', keywords (e.g., https://wiki.openstack.org/) as internal links of the OpenStack project. We also consider the links under "https://blueprints.launchpad.net/openstack" and "https://github.com/openstack" as the internal links of the OpenStack project. Similarly, for Qt, we consider the links with the domain name that has the 'qt' keyword as internal links of the Qt project. We also consider the links under "http://github.com/qt/" as internal links of Qt. Links that are not identified as internal links will be identified as external links. Once we identify whether the links are internal or external, we examine the popular domain for internal and external links. To do so, we measure the frequency of links in each domain.

Link Target Types To understand what kinds of link targets are referenced in review discussions, we perform a manual analysis on a statistically representative sample of our link dataset.

- (I) Representative dataset construction. As the full set of our constructed data is too large to manually examine their link targets, we then draw a statistically representative sample. The required sample size was calculated so that our conclusions about the ratio of links with a specific characteristic would generalize to all links in the same bucket with a confidence level of 95% and a confidence interval of 5.[7] The calculation of statistically significant sample sizes based on population size, confidence interval, and confidence level is well established (Krejcie and Morgan 1970). We randomly sample 379 internal links and 327 external links from the unique links of the OpenStack project, and 363 internal links and 309 external links of the Qt project. To remove the threat of links being inaccessible (404), our approach includes verifying each link until our sample size is reached. To do so, we first randomly select 500 internal candidate links and 500 external candidate links for each studied project. Then we automatically verified and filtered out links that are inaccessible. In the end, we filtered out 70 inaccessible internal links and 138 inaccessible external links for the OpenStack project, and 75 inaccessible internal links and 147 inaccessible external links for the Qt project.
- (II) Manual coding. To classify the type of link targets, we perform two iterations of manual coding. In the first iteration, the first two authors independently coded 50 internal and external links in the sample. The initial codes were based on the coding scheme of Hata et al. (2019) which provides the 14 types of link targets in source code comments. However, we found that their codes did not cover all link targets in our datasets. Hence, the following five codes emerged from our manual analysis in the first iteration:
  - *Communication channel:* links target for the mailing list, chat room.
  - *GitHub activity:* links target for pull requests, commits, and issues.

[7] https://www.surveysystem.com/sscalc.htm

- *Media:* links target for pictures and videos.
- *Memo:* links target for the personally recorded documentation.
- *Review:* links target for the code review.

To validate our codes, we perform a second iteration of manual coding. In this iteration, the three authors of this paper independently coded another 30 internal and external links in the sample. Then, we measure the inter-rater agreement using Cohen's Kappa across the 19 types of link targets. The score of the Kappa agreement is 0.83, which is implied as nearly perfect (Viera et al. 2005). Based on this encouraging result, we divided the remaining samples into two sets. Then, the first author independently coded the first set and the second author independently coded the second set.

3.4 RQ2 Analysis

To answer RQ2: Does the number of links shared in the review discussion correlate with review time?, we perform a statistical analysis using a non-linear regression model to investigate the correlation between the link sharing and the review process (i.e., review time), while consider several confounding variables. Similar to the prior studies (Thongtanunam et al. 2017; Ruangwan et al. 2018), the goal of our statistical analysis is *not* to predict the review time, but to understand the associations between the link sharing and the review time. In this section, we first describe our selected explanatory variables, then we describe our model construction, and finally, we explain how we analyze the model. Figure 3 presents an overview of our RQ2 quantitative analysis.

Explanatory Variables Table 2 describes the 14 metrics that are used as explanatory variables in our model. Since we want to investigate the correlation between link sharing and the code review time, we count the number of internal links, external links, and the total links. As prior studies have shown that several factors can have an impact on the review time (Kononenko et al. 2018; Thongtanunam et al. 2016), we also include 11 variables shown in Table 2 into our model. Similar to the prior work (Thongtanunam et al. 2017; McIntosh et al. 2014), 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.

For the dependent variable (i.e., review time), we measure the time interval in hours from the time when the first comment was posted until the time when the last comment was posted. We did not include the time between the patch was submitted and the first comment was made because this period of time could be the review waiting time of a patch, which is not a key focus of this study.

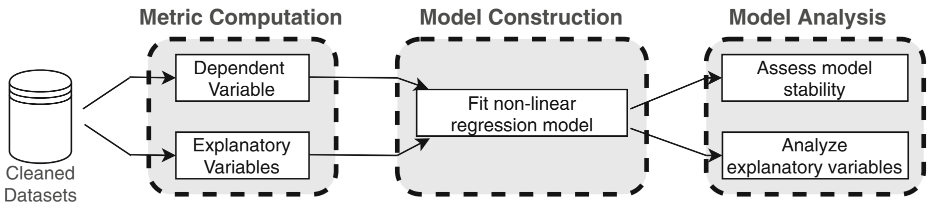


Fig. 3 An overview of our RQ2 quantitative analysis

Table 2 The studied explanatory variables

Confounding variables	Description
Add	The number of added lines by a patch.
Delete	The number of deleted lines by a patch.
Patch size	The total number of added and deleted lines by a patch.
Purpose	The purpose of a patch, i.e., bug, document, feature.
# Files	The number of files what were changed by a patch.
# Revisions	The number of review iterations.
Patch author experience	The number of prior patches that were submitted by the patch author.
# Comments	The number of messages posted in a review discussion by reviewers and the patch authors, excluding messages for change updates and the number of inline comments.
# Author comments	The number of messages posted in a review discussion by the patch author, excluding messages for change updates and the number of inline comments.
# Reviewer comments	The number of messages posted in a review discussion by reviewers, excluding messages for change updates and the number of inline comments.
# Reviewers	The number of developers who posted a comment to a review discussion.
Link sharing variables	Description
# External links	The number of external links shared in the general discussion.
# Internal links	The number of internal links shared in the general discussion.
# Total links	The number of internal and external links shared in the general discussion.

Model Construction (MC) To investigate the association between link sharing and review time, we choose the Ordinary Least Squares (OLS) multiple regression model. This technique allows us to fit the nonlinear relationship between the explanatory variables and the dependent variable. We adopt the model construction approach of Harrell et al. (1984), which was also used by Mcintosh et al. (2016). This construction approach also enables a more accurate and robust fit of the dataset, while carefully considering the potential for over-fitting. The model construction approach consists of five steps and we explain below:

(MC1) Estimate budget for degrees of freedom. As suggested by Harrell et al. (1984), we estimate a budget for the model before fitting our regression model, i.e., the maximum number of degrees of freedom that we can spend. We spend no more than $\frac{n}{15}$ degrees of freedom in our OLS model, where n refers to the number of studied reviews in the dataset.

(MC2) Normality adjustment. OLS expects that the response variable (i.e., review time) is normally distributed. Since software engineering data is often skewed, we analyze the distribution of review time in each studied project before fitting our model. We use `skewness` and `kurtosis` function of the `moments` R package to check whether or not our modeled dataset is skewed. If the distribution of review time is skewed (i.e., p-value < 0.05), similar to prior work (Mcintosh et al. 2016), we use a log transfor mation to lessen the skew in order to better fit the assumption of the OLS technique.

(MC3) Correlation and redundancy analysis. Highly correlated explanatory variables can interfere with each other when examining the significance of the relationship between each explanatory variable and the response variable, which potentially leads to spurious conclusions. Hence, we use the Spearman rank correlation (ρ) to assess the correlation between each pair of metrics. We repeat this process until the Spearman correlation coefficient values of all pairs of metrics are less than 0.7. Although correlation analysis reduces collinearity among the explanatory variables, it may not detect redundant variables (i.e., an explanatory variable that does not have a unique signal from other explanatory variables). To assure that studied variables provide a unique signal, we use the `redun` function of the `rms` R package to detect the redundant variables and remove them from our models.

(MC4) Allocating degrees of freedom. After removing highly correlated and redundant variables, we consider how to allocate degrees of freedom to the remaining variables most effectively. Similar to prior work (Mcintosh et al. 2016), we use the `spearman2` function of the `rms` R package to calculate the Spearman multiple ρ^2 between the explanatory and response variables. The larger Spearman multiple ρ^2 denotes to the higher potential of sharing a nonlinear relationship. Thus, variables with larger ρ^2 values are allocated more degrees of freedom than variables with smaller ρ^2 values. To avoid the over-fitting issue, we only allocate three to five degrees of freedom to those variables with high ρ^2 values and allocate one degree of freedom (i.e., a linear relationship) to variables with low ρ^2 values.

(MC5) Fitting statistical models. Once we decide the allocation of freedom degrees to the variables, we construct a non-linear multiple regression model. Similar to prior work (Mcintosh et al. 2016), we use restricted cubic splines which force the tails of the first and last degrees of freedom to be linear, to fit our modeled dataset. We use the `rcs` function of the `rms` R package to assign the allocated degress of freedom to each explanatory variable. Then, we use the `ols` function of the `rms` R package to construct the model.

Model Analysis (MA) After the model construction, we assess the goodness of fitting our models and examine the relationship between the review time and the explanatory variables especially for the number of internal and external links shared in the review discussions. We analyze the model using the three steps: (1) assessing the goodness of fit and model stability, (2) estimating the power of explanatory variables, and (3) examining the relationship between the explanatory variables and the review time. We describe each step in detail below:

(MA1) Assessing model stability. We use the adjusted R^2 (Hastie et al. 2009) to evaluate how well the model fits the dataset based on the studied metrics. However, the adjusted R^2 can be overestimated if the model is overfit to the dataset. Hence, we use the bootstrap validation approach to estimate the optimism of the adjusted R^2. To do so, we first generate a bootstrap sample, i.e., a sample with replacement from the original dataset. Then, we construct a model using the bootstrap sample (i.e., a bootstrap model). The optimism is a difference in the adjusted R^2 values between the bootstrap model when applied to the original R^2 optimism. Finally, we subtract the average R^2 optimism from the initial adjusted R^2 value to obtain the optimism-reduced adjusted R^2.

(MA2) Estimating the power of explanatory variables. To identify the variables that are highly correlated with the review time, we estimate the power of explanatory variables

that contribute to the fit of our model. Similar to prior work (Mcintosh et al. 2016), we use Wald χ^2 maximum likelihood tests to jointly test a set of model terms for each explanatory variable since these variables are allocated more than one degrees of freedom. The larger the χ^2 of an explanatory variable is, the larger the contribution that the variable made to the model. We use the `anova` function of the `rms` R package to report both the Wald χ^2 value and its corresponding p-value.
(MA3) Examining relationship. Finally, we examine the direction of the relationship between each explanatory variable and the review time. To do so, we use the `Predict` function of the `rms` package to plot the estimated review time while varying the value of a particular explanatory variable and hold the other explanatory variables at their median values.

3.5 RQ3 Analysis

To answer RQ3: What are the common intentions of links shared in the review discussion?, we conduct a qualitative analysis to investigate the intention of link sharing. In particular, we perform manual coding on a representative sample. Note that we use the same representative sample used in RQ1 (see Section 3.3). Below, we describe our coding scheme and manual coding process.

Coding scheme of intentions for link sharing We hypothesis that links are shared to fulfill different information needs. Hence, we use the taxonomy of information needs of Pascarella et al. (2018) as our initial coding scheme. We rely on their taxonomy because their taxonomy is closely relevant to our study, i.e., what kinds of information that was requested by reviewers during code review in the MCR context, and the taxonomy has been validated based on a semi-structured interview.

To test how well the taxonomy of information needs of Pascarella et al. (2018) can be used to classify the intentions of link sharing, we randomly select 50 samples from our representative datasets and classify them into the taxonomy of information needs. More specifically, we identify the category of information needs which the share link aims to fulfill. This classification is conducted by the two authors of this paper. After the classification, the first four authors discuss whether the taxonomy of information needs can be used. We find that the links can be classified into the taxonomy of the information needs of Pascarella et al. However, we refine the taxonomy to focus on the intentions of link sharing since the taxonomy of Pascarella et al. focuses on the reviewers' questions. Table 3 shows the refined taxonomy of intentions of our work, which is derived from the taxonomy of information needs of Pascarella et al.

Manual coding process After we refine the taxonomy of intentions for link sharing, we validate our coding schema by classifying another 30 links of the representative samples based on our taxonomy. This coding was conducted by the three authors of this paper. Then, we measure the inter-rater agreement using Cohen's Kappa. The average Cohen's Kappa score is 0.72 which indicates "substantial agreement" (Viera et al. 2005). The somewhat lower agreement can be explained by the need to extrapolate the intention behind a link from its context in the review discussion alone, without being able to interview the developer who added the link. After the validation, we splitted the remaining links into two sets. Then, the first author independently coded the first set and the second author independently coded the second set. In total, we manually classify 1,378 links. Note that when we classify the

Table 3 The taxonomy of intentions for sharing links

Category	Description	Taxonomy of information needs (Pascarella et al. 2018)
Providing Context	The link is shared to provide the additional information related to the implementation.	Context–Reviewers ask about the information aimed at clarifying the context of a given implementation
Elaborating	The link is shared to complete the information or references related to the patch.	Rationale–Reviewers ask questions to get a rationale why the patch was implemented in a certain way.
Clarifying	The link is shared to clarify some doubts about the review process or to correct the reviewer's understanding of the patch.	Correct Understanding–Reviewers ask questions to confirm the reviewer's interpretation/understanding or to clarify doubts.
Explaining Necessity	The link is shared to inform more suitable solutions or explain the reasons why the patch is no longer needed.	Necessity–Reviewers need to know whether the patch (or a part of it) is necessary.
Proposing Improvement	The link is shared to point out an alternative solution or suggestion improvement.	Suitability of An Alternative Solution–Reviewers pose a question to discuss options and alternative solutions to the implementation of the patch.
Suggesting Experts	The link is shared to point out to an expert (other developers) who should be involved.	Specialized Expertise–Reviewers ask other reviewers to contribute with their specialized expertise.
Informing Splitted Patches	The link is shared to inform that the patch has been splitted.	Splittable–Reviewers ask questions to seek the possibility of splitting the patch into multiple, separated patches.

links, we also consider the textual content of the comments that contain links and the entire discussion thread to have a better understanding of the context.

4 Case Study Results

In this section, we present the results for each of our research questions.

4.1 RQ1: To what extent do developers share links in the review discussion?

To answer RQ1, we analyze (1) the trend of link sharing (i.e., how often reviews have shared links overtime), (2) the common domains of the shared links, and (3) the types of link targets. Figures 4, 5, and Table 4 show the results of our analysis which is described in Section 3.3. We now discuss our results below.

Link Sharing Trend In 2015–2019, 25% and 20% of the reviews have at least one link shared in a review discussion within the OpenStack and Qt, respectively. Figure 4 presents the proportion of reviews that have at least one link in the review discussion over time. We find that the proportion of reviews has an increasing trend from 2011 until 2014 for both OpenStack and Qt. Then the proportion of reviews remains at 20%–30% (OpenStack) and 15%–20% (Qt) from 2014 and onwards. This result suggests that links are commonly shared in a review discussion for code review.

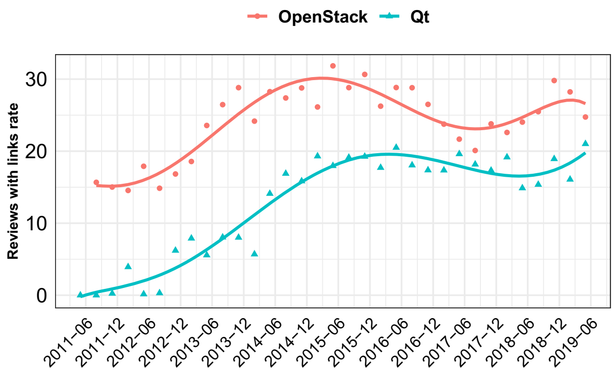


Fig. 4 The proportion of reviews that have links in an interval of three months. In 2015-2019, 25% and 20% of the reviews have at least one link shared in a review discussion within the OpenStack and Qt

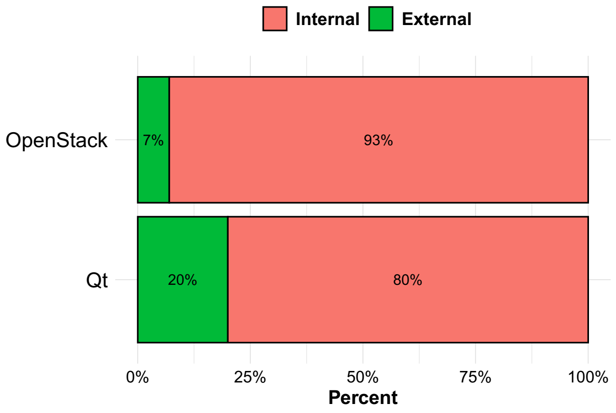


Fig. 5 The proportion of internal and external links. 93% and 80% of links that are shared in code reviews are internal links within the OpenStack and Qt

Table 4 The five most common domains in OpenStack and Qt. review.openstack.org and codereview.qt-project.org are the most common domains within the OpenStack and Qt

	Internal		External	
	OpenStack	Qt	OpenStack	Qt
Top 1	review.openstack.org (51%)	codereview.qt-project.org (79%)	github.com (15%)	paste.kde.org (14%)
Top 2	github.com/openstack (13%)	bugreports.qt.io (6%)	docs.python.org (4%)	github.com (6%)
Top 3	bugs.launchpad.net (7%)	testresults.qt.io (4%)	gist.github.com (4%)	msdn.microsoft.com (5%)
Top 4	logs.openstack.org (6%)	doc.qt.io (2%)	bugzilla.redhat.com (3%)	pastebin.kde.org (3%)
Top 5	wiki.openstack.org (4%)	wiki.qt.io (2%)	stackoverflow.com (2%)	gcc.gnu.org (3%)

Common Domain in the Project 93% and 80% of links that are shared in code reviews are internal links. Table 4 shows the ratio between internal and external links that are shared in OpenStack and Qt. We find the majority of links that are shared in the code reviews are internal links (i.e., links that are directly related to the projects). More specifically, Table 4 shows that 93% of links shared in OpenStack reviews are internal links, while only 7% of the links are external links (i.e., not directly related to OpenStack). Qt also has a similar ratio, where 80% of the links shared in Qt reviews are internal, and 20% of the links are external. These results indicate that links that are often shared in the review discussion are directly related to the project. In addition, Table 4 shows that the most common domains for the internal links are review.openstack.org and codereview.qt-project.org, which account for 51% and 79% of the internal links shared in OpenStack and Qt, respectively. The other common domains of the internal links shared in OpenStack reviews are github.com/openstack (OpenStack mirror projects in GitHub), bugs.launchpad.net, log.openstack.org, and wiki.openstack.org, which account for 30% of the internal links. For Qt, the other four common domains of the internal links are bugreports.qt.io, testresult.qt.io, doc.qt.io, and wiki.qt.io, which account for 14% of the internal links. On the other hand, the most common domain of external links is github.com for OpenStack and paste.kde.org for Qt.

Link Target Types We now further examine what kinds of information to which the shared links are referenced. Based on a manual coding on a statistical representative sample, Table 5 shows that reviews (a set of code changes) are the most frequently referenced by

Table 5 Frequency of link target types in our representative samples

	Internal		External	
	OpenStack	Qt	OpenStack	Qt
Licence	–	–	0.3%	–
Software homepage	1.1%	0.6%	9.2%	3.6%
Specification	2.6%	–	2.8%	0.6%
Organization homepage	–	–	0.6%	0.3%
Tutorial or article	7.1%	5.8%	18.7%	14.6%
API documentation	–	2.5%	15.3%	16.5%
Blog post	–	–	2.8%	1.9%
Bug report	9.2%	9.9%	8.3%	8.4%
Research paper	–	–	0.3%	–
Code	13.5%	4.1%	13.5%	10.4%
Forum thread	–	0.8%	0.3%	0.6%
Book content	–	–	0.6%	1.0%
Q&A thread	–	–	1.2%	0.6%
Stack Overflow	–	–	3.1%	1.9%
Communication channel	2.6%	0.3%	4.9%	3.6%
GitHub activity	0.3%	–	4.9%	3.6%
Media	–	–	2.1%	5.5%
Memo	5.8%	1.1%	5.8%	6.5%
Review	55.9%	73.6%	–	0.3%
Others	1.8%	1.4%	5.5%	20.1%

The bold target categories are complemented from the work by Hata et al. (2019)

internal links, which account for 55.9% and 73.6% of the internal links for OpenStack and Qt, respectively. The other kinds of information that are frequently referenced by the internal links are bug report (9.2% for OpenStack; 9.9% for Qt), source code (13.5% for OpenStack; 4.1% for Qt), and tutorial or article (7.1% for OpenStack; 5.8% for Qt). On the other hands, we find that tutorial or article and API documentation are the two most frequent targets referenced by external links, which account for 18.7% and 15.3%, and 14.6% and 16.5% the external links shared in OpenStack and Qt, respectively. The other kinds of information that are frequently referenced by the external links are source code (13.5% for OpenStack; 10.4% for Qt), and bug report (8.3% for OpenStack; 8.4% for Qt). This might suggest that the external links often reference to temporary information. Specifically, within the Qt, we observe that 20.1% of external links are classified as Others. Through the manual analysis, these links are mostly referred to the links that can be accessible but requiring authentication. For example, when we click on the external link https://paste.kde.org/pgjaet12z, the web page shows that we need to sign in or sign up before continuing.

RQ1 Summary: In the past five years, 25% and 20% of the reviews have at least one link shared in a review discussion within the OpenStack and Qt. 93% and 80% of shared links are the internal links that are directly related to the project. Importantly, although the majority of the internal links are referencing to reviews, we find that the links referencing to bug reports and source code are also shared in review discussions. In addition, we find that the common target types of external links are tutorial and API documentation.

4.2 RQ2: Does the number of links shared in the review discussion correlate with review time?

To answer RQ2, we analyze the correlation between link sharing variables and the review time, using a non-linear regression model. Figure 6 and Table 6 show the results of our

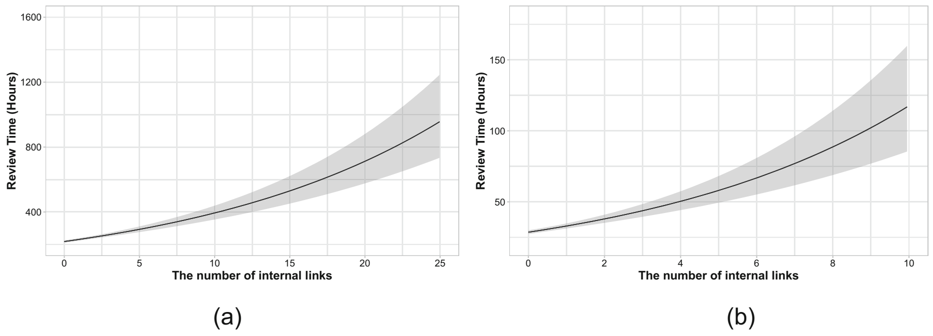


Fig. 6 The direction of the relationships between the number of internal links and the code review time. The light grey area shows the 95% confidence interval. It shows that the more internal links are shared during the discussion, the longer review time will be taken

Table 6 Review time model statistics

		OpenStack		Qt	
Adjusted R^2		0.3737		0.4580	
Optimism-reduced adjusted R^2		0.3733		0.4573	
Overall Wald χ^2		34,649		34,358	
Budgeted Degrees of Freedom		3,873		2,711	
Spent Degrees of Freedom		24		24	
Confounding variables		Overall	Nonlinear	Overall	Nonlinear
Patch size	D.F.	†		†	
	χ^2				
Add	D.F.	2	1	2	1
	χ^2	154$***$	153$***$	337$***$	337$***$
Delete	D.F.	1	–	1	–
	χ^2	1^o		0.53^o	
Purpose	D.F.	2	–	2	–
	χ^2	276$***$		131$***$	
# Files	D.F.	2	–	1	–
	χ^2	31$***$		0.34^o	
Patch author Exp.	D.F.	1	–	1	–
	χ^2	220$***$		98$***$	
# Comments	D.F.	†		†	
	χ^2				
# Author comments	D.F.	3	2	3	2
	χ^2	1936$***$	1679$***$	2216$***$	1549$***$
# Reviewer comments	D.F.	4	3	3	2
	χ^2	1360$***$	1066$***$	4908$***$	4036$***$
# Reviewers	D.F.	†		†	
	χ^2				
# Revisions	D.F.	3	2	3	2
	χ^2	3237$***$	1847$***$	2038$***$	1687$***$
Link sharing variables		Overall	Nonlinear	Overall	Nonlinear
# External links	D.F.	1	–	1	–
	χ^2	3^o		0.22^o	
# Internal links	D.F.	1	–	1	–
	χ^2	**119**$***$		**78**$***$	
# Total links	D.F.	†		†	
	χ^2				

Among link sharing variables, # Internal links has a significant correlation with review time, while # External links does not have

†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:

o $p \geq 0.05$; $*$ $p<0.05$; $**$ $p<0.01$; $***$ $p<0.001$

model construction and model analysis which is described in Section 3.4. We now discuss our results below.

Model Construction We first describe our model construction. Table 6 shows the surviving explanatory variables that are used in our models. Based on the hierarchical clustering analysis, we remove those explanatory variables that are highly correlated with one another, i.e., Patch size, # Comments, # Reviewers, and # Total links. For the surviving explanatory variables, we do not find a redundant variable, i.e., the variable that has a fit with an R^2 greater than 0.9 during the redundancy analysis. The budgeted degrees of freedom are then carefully allocated to the surviving explanatory variables based on their potential for sharing a nonlinear relationship with the response variable as described in Section 3.4. We spent 24 degrees of freedom on OpenStack and Qt models.

Model Analysis We now analyze the goodness of fit of our models. Table 6 shows that our non-linear regression model achieve an adjusted R^2 of 0.3737 (OpenStack) and 0.4580 (Qt). The adjusted R^2 scores are acceptable as our models are supposed to be explanatory not for the predictive purpose (Moksony 1999). Taking overestimation into account, after applying the bootstrap techniques with 1,000 iterations, we find that the optimism of an adjusted R^2 is 0.0004 and 0.0007 for OpenStack and Qt models, respectively. The result indicates that our constructed models are stable and can provide a meaningful and robust amount of explanatory power.

We now discuss the explanatory power of the variables of interests (i.e., # External links, # Internal links) and their relationship with the review time. Table 6 shows the explanatory power (Wald χ^2 value) of our explanatory variables that contribute to the fit of our models. In the table, the 'Overall' column shows the Wald χ^2 value of the entire model fit that the explanatory variable contributes to the fit of the model, while the 'Nonlinear' column shows the Wald χ^2 value that the nonlinear component of the explanatory variable contributes to the fit of the model. Taking a look into link sharing variables, the statistics show that there is no significant correlation between the number of external links and the review time (*p-value* >0.05) for both studied projects. On the other hand, we observe that the number of internal links has a significant correlation with the review time (*p-value* <0.001). However, the explanatory power of the number of internal links is not as large as the explanatory powers of the confounding variables. More specifically, the Wald χ^2 values of the number of internal links account for only 0.4% ($\frac{119}{34,649}$; OpenStack) and 0.3% ($\frac{78}{34,358}$; Qt), while the Wald χ^2 values of the variables range from 0.4%–10% (OpenStack) and 0.3%–13% (Qt). These results suggest that the internal link has a relatively weak correlation with the review time when compared to other variables.

Figure 6 shows the relationship between the number of internal links and the review time. We find that for both studied projects, the number of internal links has an increasing relationship with the review time. In other words, the more internal links shared in the review discussions, it tends to take a longer time for the patch to be reviewed. There are two possible conjectures that how internal links may contribute in a longer review process. One potential reason is that developers may spend time on finding the related internal links. Another possible reason is that since the internal links are closely related to the project resources, it will take time for developers to pay attention to and understand the project environment. Although the model shows an increasing relationship, we can not explain the causality between the internal link count and the review time, since before the internal links occur, the long review time could have already been taken.

RQ2 Summary: Our non-linear regression models show that the internal link has a significant correlation (but relatively weak) with the review time. However, the external link is not significantly correlated with the review time. Furthermore, we observe that the number of internal links has an increasing relationship with the review time.

4.3 RQ3: What are the common intentions of links shared in the review discussion?

To answer RQ3, we analyze (1) the kinds of common intentions of sharing links, and (2) the frequency of these intentions within our studied projects. Below, we first provide representative examples for each intention type, and then we discuss the result of the frequency of intentions (Fig. 7).

Taxonomy of Common Intentions Seven intentions of sharing links are classified through our qualitative analysis, which is described in Section 3.5:

(I) Providing Context. This category emerges by grouping discussions in which the links are shared to provide additional information related to the implementation. The Ex 1,[8] shows that the reviewer shared an internal review link for the author to inform the review team of the dependent patch. During the classification process, we observe that apart from sharing review links, the developers also share specific log results or screenshots for review teams to better understand the code change implementation. For instance, in the Ex 2[9] the reviewer self came across a test failure. To reduce the confusion, the reviewer attached an external memo link recorded with the log result for the review team to check. Note that although the log result is from the CI tool of the studied project, the link that is shared is not directly related to the project.

Ex 1
Reviewer: Depends on [internal review link].

Ex 2
Reviewer: Hmm, test failures from Jenkins look real, if confusing: [external memo link].

(II) Elaborating. The category refers to the links that are shared to complete the information provided in the review comment or references related to the patch. In this case, the keywords are usually left on the comments such as "refer to", "for example", "for your information". We show the following two representative examples to describe the intention of elaborating regarding internal and external links. In the Ex 3,[10] the reviewer presented his review suggestion, and to complete the opinion, the reviewer as well shared an internal code link as a reference for the author. For the external link, as shown in the Ex 4,[11] the author shared a Q&A thread link to explain what is the needed HZ value.

[8]https://codereview.qt-project.org/c/qt/qtbase/+/194731
[9]https://review.opendev.org/#/c/25515/
[10]https://review.opendev.org/#/c/75839/
[11]https://review.opendev.org/#/c/236210/

Ex 3
Reviewer: I would prefer that you didn't merge this. Like mentioned in previous review, if 'import' is removed and there is no code/comment/docstring, the license header should be removed as well. Please refer to: [internal doc link]

Ex 4
Author: For example on page [external Q&A thread link] it is well explained for what is this HZ value needed there.

(III) Clarifying. In this category, the links are shared to clarify some doubts of review process or to correct the reviewer's understanding of the patch. We find the clarification can be claimed from either the reviewer or the patch author aspect. The Ex 5,[12] illustrates the case where the patch author used an internal code link to address the reviewer's doubts about the undefined behaviour with the code change. In the Ex 6[13] the patch author thought that signed types implicitly converting to unsigned ones was not a problem. However, the reviewer shared external doc links to explain that it is a true problem that should be focused on.

Ex 5
Reviewer: What is this fixing? Is there an undefined behaviour i am not seeing?
Author: I probably need to add this to the commit message, but: 1. by fixing it to QObject, we fix it to a specified size, independent of which class type the member belonged to (see long explanation in [internal review link].) 2. since it no longer depends on the class type, the impl() functions are generated based only on the signal and slot's arguments, not on the class they are from. This reduces template bloat. [internal code link].

Ex 6
Author: Signed types implicitly convert to unsigned ones. That's not a problem.
Reviewer: I think that *is* a problem: going from signed to unsigned is not an integral promotion [external doc link], but an integral conversion [external doc link]

(IV) Explaining Necessity. This category refers to the links that are shared to inform more suitable solutions or explain the reasons why the patch is no longer needed. We observe this category can happen in both internal and external links. For example, as shown in the Ex 7,[14] the reviewer shared a review link and pointed out that the linked review had already changed network_type validation using a simple approach. And further suggested that the current patch was not really needed. The Ex 8[15] is related to the shared external link with the intention of explaining necessity. One developer considered the submitted change was not sufficient since libproxy already had a fix. At the end of the comment, the developer shared an external GitHub link to indeed prove that the fix has been done through this link. Note that the external GitHub link is not directly related to the GitHub of the studied project.

[12]https://codereview.qt-project.org/c/qt/qtbase/+/160883
[13]https://codereview.qt-project.org/c/qt/qtbase/+/186305
[14]https://review.opendev.org/#/c/22928/
[15]https://codereview.qt-project.org/c/qt/qtbase/+/200463

Ex 7
Reviewer: I would prefer that you didn't merge this
The WIP ml2 patch at [internal review link] changes the network_type validation in the core to simply validate a string. Is anything more really needed?

Ex 8
Author: This change is insufficient. libproxy already has a mutex, so adding another one won't solve anything.
* [external GitHub link]

(V) Proposing Improvement. In this category, we group discussions in which the links are shared to point out an alternative solution or suggestion improvement. As shown in the Ex 9,[16] the reviewer provided the author with an internal GitHub code link and suggested the author should follow the proposed method. Similarly, the reviewer asked the author to do something like what the shared long-term memory link did in the Ex 10.[17]

Ex 9
Reviewer: The compute api loading code and the new hostapi loading code (and network api loading code for that matter) should follow the method that the volume api loading code uses here: [internal GitHub code link]

Ex 10
Reviewer: could we do something like that: [external long-term memory link]
... a bit rough but you should get the idea ... get all (without specifying any attr) layer

(VI) Suggesting Experts. We define this category as the links are shared to point out to an expert (other developers) who should be involved. In Ex 11,[18] the author shared an internal review link related to the spec change and invited the reviewer named John to have a review as well. In the Ex 12,[19] in order to address the reviewer's question, the author used @ to suggest an expert along with an issue link to point out he was an experienced maintainer regarding such a situation.

Ex 11
Author: Hi John, spec is on review: [internal review link]. Could you please review it?

Ex 12
Reviewer: Or is it a bug in the protocol?
Author: @psychon (a.k.a Uli Schlachter) who is a maintainer of Awesome WM has a good comment regarding the situation with the standard [external GitHub issue link].

[16]https://review.opendev.org/#/c/12311
[17]https://codereview.qt-project.org/c/qt/qtbase/+/126975
[18]https://review.opendev.org/#/c/219248/
[19]https://review.opendev.org/#/c/219248/

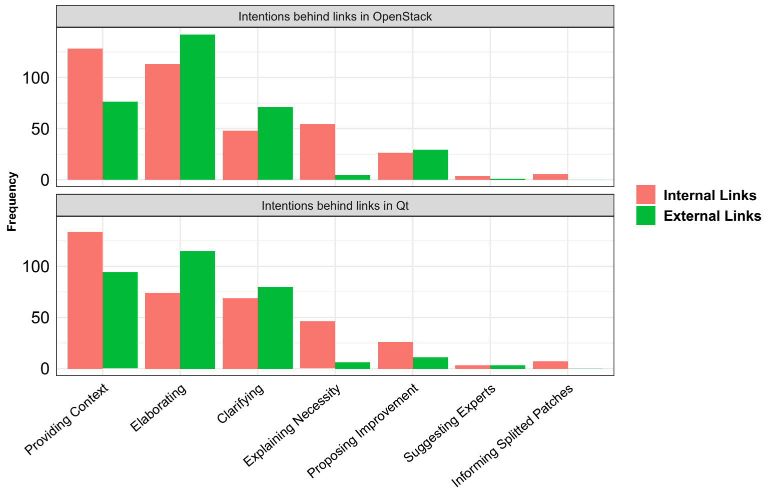


Fig. 7 Distribution of seven intentions behind sharing links across the studied projects. The results show that *Providing Context* and *Elaborating* are the most common intentions for internal and external links, respectively

(VII) Informing Splitted Patches. In this category, the links are shared to inform that the patch has been splitted. We only find this intention existing behind the internal link sharing. As shown in the Ex 13,[20] the reviewer was trying to abandon the current patch and split it into several small chunks to fulfill the request, along with internal review links.

Ex 13
Reviewer: I'm abandoning this patch as a result of the request to split this review into smaller chunks - these reviews are [internal review link].

Frequency of intentions for link sharing We now examine what are the common intentions for sharing links in the review discussion. Figure 7 shows the distribution of each intention category within internal and external types for OpenStack and Qt. The figure clearly reveals that not all the intentions are equally distributed and highlights the presence of a particular type. Specifically, for internal links (i.e., directly related to the project), we observe that *Providing Context* category is the most frequent intention in OpenStack and Qt (128 and 134 links, respectively). The result indicates that internal links are commonly shared in the review discussion to provide additional information related to the implementation of a patch. For the external links (i.e., not directly related to the project), we find that the most common intention is *Elaborating* in both projects (145 and 115 links, respectively). This finding suggests that external links are commonly shared to complement the information provided in the review comment. In addition, we observe that (i.e., *Informing Splitted Patches* and *Suggesting Experts*) are the least common intentions for link sharing. This observation suggests that links may not be able to fulfill these information needs.

Upon closer inspection on the review links (i.e., the most common shared link target in RQ1), as shown in Table 7, we find that the most frequent intention is to provide context,

[20]https://review.opendev.org/#/c/103167

Table 7 The three most frequent intentions of sharing review links

Intention	OpenStack	Qt
Providing Context	40%	40%
Explaining Necessity	23%	18%
Elaborating	19%	16%

accounting for 40% for both OpenStack and Qt project. Through our manual coding, we observe that such context is usually concerning the patch dependency[21] and integration test environment.[22] The second most frequent intention of sharing review links is to explain necessity, accounting for 23% and 18% for the OpenStack and Qt project, respectively. The third most frequent intention is to elaborate, i.e., 19% and 16% for the OpenStack and Qt project, respectively.

RQ3 Summary: We identify seven intentions of sharing links: (1) Providing Context, (2) Elaborating, (3) Clarifying, (4) Explaining Necessity, (5) Proposing Improvement, (6) Suggesting Experts, and (7) Informing Splitted Request. We find that providing context is the most common intention for sharing internal links and elaborating (i.e., providing a reference or related information) to complement review comments is the most common intention for sharing external links.

5 Discussions

In this section, we discuss the implications of our analysis results. To gain a further insight of the perception of sharing links by developers, we conduct a survey study with OpenStack and Qt developers. Below, we first present the results of our survey. Then we provide suggestions to the patch author, review teams, and researchers.

5.1 Developer Feedback

To gain insights into developer perception of link sharing, we sent out a survey[23] to OpenStack and Qt developers, with the goal to (i) receive feedback on the three study findings, (ii) solicit developers' opinions, and (iii) collect insights into the developer experience with existing functionalities (i.e., related changes, same topic). The survey consists of two likert scale questions and five open-ended questions. We sent our online survey invitation to 1,871 developers who have shared links in the past based on our studied dataset. The survey was open from January 22 to February 13, 2021. We received responses from 53 developers in total. To analyze the responses of the open-ended questions, we use the card sorting method. Below, we present our survey questions and discuss the survey results.

Feedback on Findings of RQ1 and RQ2 Table 8 shows the feedback on our findings of RQ1 and RQ2. For the finding of RQ1, forty-eight respondents (90% = 48/53) agreed that

[21]https://review.opendev.org/#/c/612393/
[22]https://review.opendev.org/#/c/453537/
[23]https://forms.gle/hiBameBdGFMhnxNSA

Table 8 Feedback on findings of RQ1 and RQ2, using the Likert-scale scale below: 1 = Strongly disagree, 2 = Partially disagree, 3 = No opinion, 4 = Partially agree, 5 = Strongly agree

	1	2	3	4	5
Finding of RQ1–"Developers often share internal links to reference reviews, bug reports and source code, while external links often reference tutorials and API documentation."	0	3	2	31	17
Finding of RQ2–"A review that has an internal link shared during its review discussion is likely to take reviewing time longer than other reviews."	3	24	15	11	0

developers often share internal links to reference reviews, bug reports, and source code, while external links often reference tutorials and API documentation. In the open-ended questions, six respondents reported that the external link target is also to point out to bug reports outside the projects. This response is consistent with our analysis results of RQ1 as well, i.e., bug report is the fourth frequent external link target (see Table 5). Interestingly, one respondent stated that "This (links) is useful for example for newcomers that may have missed guidelines or did not found the corresponding bug report when doing submitting a fix.".

We found that eleven respondents partially agreed with the finding of our RQ2, while fifteen respondents did not have opinion and twenty-seven respondents disagreed with our findings. One of the respondents who agree cited that "Sometime we are not familiar wit the exact context of internal item.". Eleven respondents who did not agree reported that the information brought with the shared links is useful and could aid the review process. For instance, one respondent cited that "Links usually provide a concise and clear answer compared to trying to explain it in prose.". Such perception of developers is consistent with our intuition as stated under the RQ2 Motivation in the Introduction. However, the analysis result shows an inverse relationship. There could be other confounding factors that play a role, and future work should further investigate the causality of this relationship.

Survey on Intentions of Sharing Links In the survey questions, we also asked the respondents to select the intention(s) that they usually use when sharing link. Table 9 shows the most selected intention of sharing link is to provide context (48/53 = 90% of respondents). The second most frequent intention from developers was to explain necessity using links (forty respondents voting), followed by the intention to elaborate (thirty-five respondents voting). These frequent intentions are overall consistent with our findings of our RQ3, especially for internal links (see Fig. 7). The least frequent intentions from respondents are to inform splitted patches (17 respondents) and suggest experts (6 respondents). In regards to the open-ended feedback, one respondent stated that "Usually when doing a code review, if an expert should take a look at the review, he will likely be added by one of the reviewer if not already by the submitter.".

Survey on Perception of Existing Functionalities of Sharing Related Patch Finally, we asked the respondents about their experience and perception of the Gerrit functionalities, which provide review links of related changes and the same topic. Out of thirty-five responses, twenty-four respondents claimed experience in using the functionalities. On the one hand, fourteen respondents acknowledged the usefulness, especially to find related

Table 9 Respondents feedback on the intentions for sharing links

Intention	Respondent Count
Providing Context	48
Explaining Necessity	40
Elaborating	35
Clarifying	33
Proposing Improvement	26
Informing Splitted Patches	17
Suggesting Experts	6
Others	3

patches or track dependencies. On the other hand, respondents did express limitations, e.g., “It seems useful for seeing what changes are submitted at a similar point in the change history, but doesn’t seem useful for finding patches that are related by content (e.g., changing the same feature) but separated by longer periods of time.”.

When we asked about the differences between existing functionalities and the practice of sharing links in the review discussion, twelve respondents acknowledged how both approaches can complement each other. For example, one of the respondents commented: “The tool can help you find information you were already looking for, but the posting of a link is a communication option to help convey the idea to other people.”.

5.2 Suggestions

Based on the our results of RQs 1-3 and the survey results, we now present our suggestions of the study.

Suggestions for Patch Authors Our RQ1 shows that the majority of shared links are directly related to the project. Figure 5 shows that 93% and 80% of shared links are internal links for OpenStack and Qt, respectively. The link targets are diverse from different locations with the complexity of projects. As shown in Table 5 from RQ1, we found eleven different target types. We suggest that in the case of well-documented projects (i.e., OpenStack and Qt), patch authors (especially for newcomers or novice developers) should read the project related guidelines to be familiar with the environment before their submission. For instance, in the review #37843,[24] one reviewer shared a link related to the Gerrit Workflow (i.e., https://wiki.openstack.org/wiki/Gerrit_Workflow) with the patch author who was newly to the project, to avoid the broken conflict. This is also supported by the responses of developers: “This (links) is useful for example for newcomers that may have missed guidelines or did not found the corresponding bug report when doing submitting a fix.”.

Through our qualitative analysis in RQ3, the observation suggests that the information brought by the shared links is helpful as an indicator for review teams to clearly understand

[24] https://review.opendev.org/#/c/37843

the patch implementation context. For one example of clarifying intention (Ex 5) shown in Section 4.3, during the review process, the reviewer got confused about the behaviour of the fix. To reduce the confusion, the author decided to improve the patch content in the commit message, along with shared links to supplement further explanation. Another example of elaborating intention (Ex 4) illustrates that to help the review team better understand what is HZ value, the patch author shared a Q&A thread link as a reference. One surveyed respondent also suggested that "It could also decrease the review time by making more clear the intent of the code change.". Inspired by these examples, we encourage patch authors to provide more information such as implementation related information via shared links during their submission, in turn to receive efficient feedback from review teams quicker and reduce the discussion confusion.

Suggestions for Review Teams Our study indicates that the practice of sharing links can fulfill various information needs. Seven intentions are classified as shown in Table 3. Our frequency analysis in RQ3 shows that the links are commonly shared to provide the context, further elaborate, and clarify doubts. Moreover, the information that is shared through links is also diverse. The example of providing context intention (Ex 2) in Section 4.3 shows that one reviewer shared an external memo link which was recorded with CI tool test failure results, in order to solve the confusion among the review team and the patch author. On the other hand, as shown in the example of explaining necessity intention (Ex 7), the reviewer noticed that a related patch already changed the network_type validation and shared this review link in the review discussions to suggest that the current patch was no longer needed. These findings show that sharing links can help developers fulfill the information needs (the challenges discovered by Pascarella et al.), which potentially saves the reviewer effort. Thus, we suggest that during the future review process, the review teams should share links to transfer the needed information for guiding the patch author and the review process, especially for the review team that does not adopt such practice. One respondent from our survey commented that "Links usually provide a concise and clear answer compared to trying to explain it in prose." Our suggestion also extends the suggestion of Pascarrella et al. that in addition to automatic change summarization, sharing links could be another method to meet the information needs. We believe that such information may help to conduct a more efficient review and also assist with mentoring new members to the review team.

Suggestions for Researchers Our RQ1 results indicate that link sharing is becoming a popular practice during review discussions in the MCR. Furthermore, Fig. 4 shows that in the last five years, around 25% and 20% of the reviews have at least one link within the OpenStack and Qt project. As the practice of linking sharing increases, new opportunities arise for researchers to develop tool support, especially to recommend related and useful links for both the patch author and review teams in order to facilitate the review process. An intelligent tool may reduce the time for developers to find the needed links. We propose the following three potential features that could be embedded in the future code review tool: First, the mechanism to automatically recommend related patches can be improved not only based on the similar change history, but also considering the patch contents. Such limitation is also pointed by the responses "It seems useful for seeing what changes are submitted at a similar point in the change history, but doesn't seem useful for finding patches that are related by content (e.g., changing the same feature) but separated by longer periods of time.". Second, a functionality to detect alternative solution patches (i.e., patches aim to achieve the same objective) is needed, since our empirical study shows that the second most frequent intention of sharing review links is to explain necessity (See Table 7). Third, a tool to recommend guideline and tutorial related link would be especially useful for novice developers and help them to be familiar with the project environment. This study also lays the groundwork for future research on the links shared in the review process to generate the structural and dynamic properties of the emergent knowledge network, aiming to enable more effective knowledge sharing within the project.

6 Threats to Validity

We now discuss threats to the validity of our empirical study.

External Validity We perform an empirical study on two projects relying on Gerrit review tools. Although OpenStack and Qt commonly used in the prior research, the observations based on this case study may not generalize to other projects or peer review settings such as the pull-based review process. However, our goal is not to build a theory that can be fit to all projects, but rather to shed light in some large open-source projects, the links being often shared in the code reviews to provide the context, elaborate to complement review comments. We only focus on the large open-source projects with distributed teams, since most of the code review activities are performed through the code review tool (the data is available). The data or communication recorded in the small or medium team may be incomplete as they can have in-person communication or using other channels to discuss, like slack. Nonetheless, additional replication studies would help to generalize our observations. Thus, in order to encourage future replication studies, our replication package is available online including the raw review datasets, manually labeled link targets and their intentions, and the script to construct the non-linear regression model.

Construct Validity We summarize two threats regarding construct validity. First, in the identification of external and internal links, we apply the keyword search to automatically split domains into external and internal types. However, cases might occur where some domains not including any indicated keywords can still belong to internal links. To reduce such bias, we manually click the domains to validate the correctness carefully.

Second, in our qualitative analysis, especially for intention classification, intentions may be miscoded due to the subjective nature of our coding approach. To mitigate this threat, we took a systematic approach to first test our comprehension with 30 samples using Kappa agreement scores by three separate individuals. Only until the Kappa score reaches more than 0.7 (i.e., 0.83 for link targets and 0.72 for link intentions), indicating that the agreement is substantial (0.61–0.80) or almost perfect (0.81–1.00), we were able to complete the rest of the sample dataset.

Internal Validity Four related threats are summarized. The first threat is concerning the link extraction. In this study, we only consider the links which are posted in the general comments. We understand that links can also be shared in the inline comments. However, the prior work (Hirao et al. 2019) pointed out that the proportions of links in the inline comments are relatively low, accounting for 18% and 10% for OpenStack and Qt project, respectively. The analysis of links shared in inline comments may provide further insights, but may not have a large impact on our findings in this paper.

The second threat is related to the results derived from the statistical models that we fitted to our data. Though we can observe the correlation between explanatory and dependent variables, the causal effects of link sharing on the review time cannot be represented. Thus, future in-depth qualitative analysis or experimental studies are needed so as to better understand the reasons and effects of link sharing impact.

The third threat is regarding our factor selection to fit the statistical models. Other factors might also influence the review time. For instance, the prior study showed that the code ownership has an impact on the review process (Thongtanunam et al. 2016). Yet, we take commonly used metrics into account similar to the work conducted by Kononenko et al. (2018). We are confident that these selected explanatory factors are appropriate to be considered and measured.

The last threat is concerning the model performance overestimation. An overfit model may exaggerate spurious relationships between explanatory and response variables. To mitigate this concern, we validate our model results using the bootstrap-calculated optimism with 1,000 iterations.

7 Related Work

Link sharing is regarded as one key practice to achieve efficient knowledge. In this section, we compare and contrast our study to previous research in two parts: first, we consider the work that analyzes the link sharing in Q&A site and GitHub, then we introduce the work that investigates the link sharing in code review settings.

7.1 Collective knowledge through links sharing

Link sharing has become an important activity in the area of software engineering, which encourages developers to exchange knowledge, increases the learning purpose, and mitigates potential issues. The value of link sharing has been widely explored in the Q&A site and GitHub. Gomez et al. (2013) found that a significant proportion of links shared on Stack Overflow (i.e., the Q&A site for professional and enthusiast programmers) are referring readers to software development innovations like libraries and tools. Ye et al. (2017) used the URLs shared in StackOverflow to generate the structural and dynamic properties of the emergent knowledge network, aiming to enable more effective knowledge sharing in the community. With the increasing growth of GitHub, 9.6 million links exist in source code comments across 25,925 repositories (Hata et al. 2019). They identified more than a dozen different kinds of link targets, with dead links, licenses, and software homepages being the most prevalent. As the survey conducted by Baltes and Diehl (2019), 40% of participants added a source code comment with a Stack Overflow link to the corresponding question or answer in GitHub projects. They analyzed how often these URLs are present in Java files and found that developers more often refer to questions, i.e., the whole thread, than to specific answers. Related to the issue linking, existing studies have demonstrated the value of such links in identifying complex bugs and duplicate issue reports. For instance, Boisselle and Adams (2015) reported that 44% of bug reports in Ubuntu are linked to indicate duplicated work. The results of Zhang et al. (2018) showed that developers tend to link issues more cross-project or cross-project over time. To ease the recovery of links, Zhang et al. (2020) proposed iLinker to address the problem of acquiring related issue knowledge so as to improve the development efficiency. Rath et al. (2018) showed that on average only 60% of the commits were linked to specific issues and proposed an approach to detect missing links between commits and issues using process and text-related features.

7.2 Link Sharing in Code Reviews

Links are also been studied in peer code review settings. Zampetti et al. (2017) investigated to what extent and for which purpose developers refer to external online resources when raising pull requests. Their results indicate that external resources are useful for developers to learn something new or to solve specific problems. In our work, we find that in addition to providing resources for learning, external links are often shared to provide context, elaborate the review comment, and to clarify doubts. Jiang et al. (2019) found that 5.25% of pull requests have links in review comments on average in ten GitHub projects. Our finding is consistent with the work of Jiang et al. (2019). In the MCR context that uses the Gerrit code review tool, we find that in the past five years, around 25% and 20% of reviews from OpenStack and Qt have links in the review discussion. However, in addition to the prevalence, we further find that the number of internal links has an increasing relationship with review time through statistical models. Hirao et al. (2019) suggested that review linkage is not uncommon in six studied software communities. They observed five types of reivew linkage, such as patch dependency, broader context, alternative solution. Our study expanded upon the work of Hirao et al. by studying not only review links but all kinds of links. Similar to their work, we also find that the internal links (of which the majority are the review links) are shared to provide implementation context. Nevertheless, our work also shows that apart from review links, other types of links (e.g., bug reports, source code in GitHub, API documentation) are also shared in the review discussions.

8 Conclusion

In this paper, we perform an empirical study on two open source projects, i.e., OpenStack and Qt, to (1) analyze to what extent do developers share links, (2) analyze the correlation between link sharing and the review time using the statistical model, and (3) investigate the common intentions of sharing links. Our results show that the majority of shared links are internal links (directly related to the project), i.e., 93% and 80% for OpenStack and Qt. We find that although the majority of the internal links are referencing to reviews, the links referencing to bug reports and source code are also shared in review discussions. Through the statistical models, our results show that the number of internal links has an increasing relationship with the review time. Regarding the intention classification, we identify seven intentions behind link sharing, with providing context and elaborating being the most common intentions for internal and external links.

Our study highlights the role that shared links play in the review discussion and the link is served as an important resource to fulfill various information needs for patch authors and review teams. The next logical step would be a deeper study of investigating the causality of these factors and understanding the reasons why it takes a longer time to complete the review. Future research directions also include the extension of a more exhaustive study that investigate the small and medium open-source projects, the in-depth analysis of link sharing practices (e.g., an impact of links shared by a patch author and reviewers on the review process), the potential for tool support, and the management of the collective knowledge within projects.

Acknowledgements This work is supported by Japanese Society for the Promotion of Science (JSPS) KAKENHI Grant Numbers JP18H04094, JP20K19774, and JP20H05706. P. Thongtanunam was partially supported by the Australian Research Council's Discovery Early Career Researcher Award (DECRA) funding scheme (DE210101091).

References

Bacchelli A, Bird C (2013) Expectations, outcomes, and challenges of modern code review. In: Proceedings of the 35th International conference on software engineering, pp 712–721

Baltes S, Diehl S (2019) Usage and attribution of stack overflow code snippets in github projects. Empir Softw Eng :1259–1295

Baysal O, Kononenko O, Holmes R, Godfrey MW (2016) Investigating technical and non-technical factors influencing modern code review. Empir Softw Eng :932–959

Boisselle V, Adams B (2015) The impact of cross-distribution bug duplicates, empirical study on debian and ubuntu. In: 2015 IEEE 15th International working conference on source code analysis and manipulation (SCAM), pp 131–140

Ebert F, Castor F, Novielli N, Serebrenik A (2019) Confusion in code reviews: Reasons, impacts, and coping strategies. In: 2019 IEEE 26th International conference on software analysis, evolution and reengineering (SANER), pp 49–60

Fagan ME (1976) Design and code inspections to reduce errors in program development. IBM Syst J 15(3):182–211

Gomez C, Cleary B, Singer L (2013) A study of innovation diffusion through link sharing on stack overflow. In: IEEE international working conference on mining software repositories, pp 81–84

Harrell F. E. Jr., Lee KL, Califf RM, Pryor DB, Rosati RA (1984) Regression modelling strategies for improved prognostic prediction. Stat Med :143–152

Hastie T, Tibshirani R, Friedman J (2009) The elements of statistical learning, data mining, inference, and prediction. Springer, Berlin

Hata H, Treude C, Kula RG, Ishio T (2019) 9.6 Million links in source code comments: purpose, evolution, and decay. In: Proceedings of the 41st international conference on software engineering, pp 1211–1221

Hirao T, McIntosh S, Ihara A, Matsumoto K (2019) The review linkage graph for code review analytics: a recovery approach and empirical study. In: Proceedings of the international symposium on the foundations of software engineering (FSE), pp 578–589

Huizinga D, Kolawa A (2007) Automated defect prevention: best practices in software management. Wiley, Hoboken

Jiang J, Cao J, Zhang L (2019) An empirical study of link sharing in review comments. In: Li Z, Jiang H, Li G, Zhou M, Li M (eds) Software engineering and methodology for emerging domains, pp 101–114

Kononenko O, Rose T, Baysal O, Godfrey M, Theisen D, de Water B (2018) Studying pull request merges: A case study of shopify's active merchant. In: 2018 IEEE/ACM 40th International conference on software engineering: software engineering in practice track (ICSE-SEIP), pp 124–133

Krejcie RV, Morgan DW (1970) Determining sample size for research activities. Educ Psychol Meas 30(3):607–610

McIntosh S, Kamei Y, Adams B, Hassan AE (2014) The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects. In: Proceedings of the 11th working conference on mining software repositories, pp 192–201

Mcintosh S, Kamei Y, Adams B, Hassan AE (2016) An empirical study of the impact of modern code review practices on software quality. Empir Softw Eng :2146–2189

Moksony F (1999) Small is beautiful: The use and interpretation of r2 in social research. Szociologiai Szemle:130–138

Pascarella L, Spadini D, Palomba F, Bruntink M, Bacchelli A (2018) Information needs in contemporary code review. In: Proceedings of the ACM Conference on computer supported cooperative work, vol 2, pp 135:1–135:27

Rath M, Rendall J, Guo JLC, Cleland-Huang J, Mäder P (2018) Traceability in the wild: Automatically augmenting incomplete trace links. In: Proceedings of the 40th International conference on software engineering, ICSE '18, pp 834–845

Rigby PC, Bird C (2013) Convergent contemporary software peer review practices. In: Proceedings of the 9th joint meeting on foundations of software engineering, pp 202–212

Rigby PC, Storey MA (2011) Understanding broadcast based peer review on open source software projects. In: Proceedings of the 33rd International conference on software engineering, pp 541–550

Ruangwan S, Thongtanunam P, Ihara A, Matsumoto K (2018) The impact of human factors on the participation decision of reviewers in modern code review. Empir Softw Eng :973–1016

Sadowski C, Söderberg E, Church L, Sipko M, Bacchelli A (2018) Modern code review: a case study at Google. In: Proceedings of the 39th International conference on software engineering: software engineering in practice track, pp 181–190

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, pp 51:1–51:11

Thongtanunam P, Hassan AE (2020) Review dynamics and their impact on software quality. IEEE Trans Softw Eng :1–1

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, pp 1039–1050

Thongtanunam P, Mcintosh S, Hassan AE, Iida H (2017) Review participation in modern code review. Empir Softw Eng 22(2):768–817

Viera AJ, Garrett JM et al (2005) Understanding interobserver agreement: The kappa statistic. Fam Med 37(5):360–363

Ye D, Xing Z, Kapre N (2017) The structure and dynamics of knowledge network in domain-specific q&a sites: A case study of stack overflow. Empirical Softw Eng :375–406

Zampetti F, Ponzanelli L, Bavota G, Mocci A, Di Penta M, Lanza M (2017) How developers document pull requests with external references. In: Proceedings of the 25th international conference on program comprehension, pp 23–33

Zhang Y, Yu Y, Wang H, Vasilescu B, Filkov V (2018) Within-ecosystem issue linking: A large-scale study of rails. In: Proceedings of the 7th international workshop on software mining, SoftwareMining, vol 2018, pp 12–19

Zhang Y, Zhou M, Mockus A, Jin Z (2019) Companies' participation in oss development - an empirical study of openstack. IEEE Trans Softw Eng 1–1

Zhang Y, Wu Y, Wang T, Wang H (2020) ilinker: a novel approach for issue knowledge acquisition in github projects. World Wide Web 23:1589–1619

Dong Wang is currently working toward the Doctor degree in Nara Institute of Science and Technology in Japan. His research interests include code review, human factors, and mining software repositories. More about his work is available online at https://dong-w.github.io/.

Tao Xiao is a Master's student at the Department of Information Science, Nara Institute of Science and Technology, Japan. He received his BSc degree in Software Engineering from Chiang Mai University, Thailand, in 2020. His main research interests are empirical software engineering, mining software repositories, natural language processing.

Patanamon Thongtanunam is an ARC DECRA awardee and a lecturer at the School of Computing and Information System, the University of Melbourne, Australia. Prior to that, she was a lecturer at the School of Computer Science, the University of Adelaide, and a research fellow of Japan Society for the Promotion of Science (JSPS). She received PhD in Information Science from Nara Institute of Science and Technology, Japan. Her research interests include empirical software engineering, mining software repositories, software quality, and human aspect. Her research has been published at top-tier software engineering venues like International Conference on Software Engineering (ICSE), Transaction of Software Engineering, and Journal of Empirical Software Engineering (EMSE). More about Patanamon and her work is available online at http://patanamon.com.

Raula Gaikovina Kula is an assistant professor at the Nara Institute of Science and Technology (NAIST), Japan. He received his Ph.D degree from NAIST in 2013 and was a Research Assistant Professor at Osaka University from Sept. 2013 till April 2017. He is active in the Software Engineering community, serving the community as a PC member for premium SE venues (i.e., ICSE, ASE, ICSME, ESEM, MSR, and so on), some as organizing committee and reviewer for journals (i.e., IEEE TSE, Spring EMSE, Elsevier IST, and JSS). His current research interests include library dependencies and security in the software ecosystem, program analysis such as code clones, and human aspects such as code reviews. Find him at https://raux.github.io/ and @augaiko on twitter.

Kenichi Matsumoto is a professor in the Graduate School of Science and Technology at Nara Institute Science and Technology, Japan. His research interests include software measurement and software process. Matsumoto has a Ph.D. in information and computer sciences from Osaka University. He is a senior member of IEEE, and a member of the IEICE, and the IPSJ. Contact him at matumoto@is.naist.jp.

Affiliations

Dong Wang[1] · **Tao Xiao**[1] · **Patanamon Thongtanunam**[2] · **Raula Gaikovina Kula**[1] · **Kenichi Matsumoto**[1]

Tao Xiao
tao.xiao.ts2@is.naist.jp

Patanamon Thongtanunam
patanamon.t@unimelb.edu.au

Raula Gaikovina Kula
raula-k@is.naist.jp

Kenichi Matsumoto
matumoto@is.naist.jp

[1] Nara Institute of Science and Technology, Ikoma, Japan

[2] The University of Melbourne, Melbourne, Australia