PAPER Special Section on Empirical Software Engineering

Understanding Developer Commenting in Code Reviews

Toshiki HIRAO[†a)], **Raula GAIKOVINA KULA**[†b)], ***Nonmembers***, **Akinori IHARA**[††c)], ***Member***, ***and*** **Kenichi MATSUMOTO**[†d)], ***Nonmember***

SUMMARY Modern code review is a well-known practice to assess the quality of software where developers discuss the quality in a web-based review tool. However, this lightweight approach may risk an inefficient review participation, especially when comments becomes either excessive (i.e., too many) or underwhelming (i.e., too few). In this study, we investigate the phenomena of reviewer commenting. Through a large-scale empirical analysis of over 1.1 million reviews from five OSS systems, we conduct an exploratory study to investigate the frequency, size, and evolution of reviewer commenting. Moreover, we also conduct a modeling study to understand the most important features that potentially drive reviewer comments. Our results find that (i) the number of comments and the number of words in the comments tend to vary among reviews and across studied systems; (ii) reviewers change their behaviours in commenting over time; and (iii) human experience and patch property aspects impact the number of comments and the number of words in the comments.

key words: *modern code review, review comments, mining software repositories, empirical study, machine learning*

1. Introduction

Code review is widely recognized as best practice for Software Quality Assurance [1]. The highly structured processes of Fagan-style reviews [2] are renown for being time-consuming in nature. Modern code review (MCR) is a lightweight process—developers informally interact with other developers and discuss patches in a web-based review tool such as Gerrit Code Review. Companies such as Microsoft, Facebook and several OSS projects have successfully adopted the MCR process [3].

However, code reviews today still have the potential for being expensive and slow [4], especially in terms of the discussion size before a final decision is made. For example, Microsoft engineers raised concerns over modern code review workflows, stating that *'current code review best practice slows us down'* [5]. Moreover, it is generally undesirable to have bloated reviewer comments in a review system [6]. In fact, the design of most modern code review systems promotes the practice of *lazy consensus* concept where reviewers reply only to the necessary content to avoid excessive comments.

Our main objective in this paper is to understand the most important perspectives on reviewer commenting. To fulfill this, we analyze the phenomena; namely, how many comments and words are involved to complete reviews, how reviewer commenting evolves over time, and what features drive reviewer comments. The goal of the work is to understand how excessive or underwhelming comments can be identified and managed.

To this end, we conduct the large-scale exploratory and modeling studies of over 1.1 million reviews across five open source projects. We form three research questions to guide those studies:

- *(RQ_1) Is there a typical number of reviewer comments before the final decisions?*
  Motivation: Towards understanding the most impactful features on reviewer commenting, we first would like to explore the phenomenon of reviewer commenting. Hence, the study investigates the extent to which the number of reviewer comments and words in the comments range across studied systems. The analysis provides with the evidence for understanding how many comments and words normally suffice to complete or manage reviews.
  Results: There is no typical number of review comments across five studied systems. Reviewers reach a decision on a review ranging up to 13 comments, with 22 words per a comment on average. Moreover, the number of comments is mostly correlated with the number of words in the comments.
- *(RQ_2) Does reviewer commenting change over the evolution of a project?*
  Motivation: The evolution of reviewer commenting is also an essential aspect to thoroughly understand how reviewers have changed their behaviours in commenting. This analysis enriches our observations of RQ1.
  Results: The number of comments tends to steadily increase over time in the largest studied system. Furthermore, the number of comments tends to mostly stabilize in other studied systems.
- *(RQ_3) What (a) patch, (b) human and (c) management features drive reviewer comments?*
  Motivation: Our main objective of this paper is to discover the most impactful features in reviewer

Manuscript received February 28, 2019.
Manuscript revised July 1, 2019.
Manuscript publicized September 11, 2019.

[†]The authors are with Nara Institute of Science and Technology, Ikoma-shi, 630–0192 Japan.
[††]The author is with Wakayama University, Wakayama-shi, 640–8510 Japan.
a) E-mail: hirao.toshiki.ho7@is.naist.jp
b) E-mail: raula-k@is.naist.jp
c) E-mail: ihara@sys.wakayama-u.ac.jp
d) E-mail: matumoto@is.naist.jp

DOI: 10.1587/transinf.2019MPP0005

commenting through a modeling study. The results (i) allow us to have a deep understanding of the key features needed to manage reviewer commenting; and (ii) provide us with considerable findings towards future code review studies one of which is a review cost estimation approach.
Results: Human experience (e.g., reviewer experience) and patch property (e.g., patch churn) features drive reviewer comments and words in the comments. Moreover, both novice authors and experienced reviewers tend to induce more comments and words, while the large and widespread modifications also have the tendency to raise more comments and words.

Our contributions are two-fold, with key implications listed in this paper. The first contribution is an exploratory study for the phenomenon of reviewer commenting. The second is the confirmation of review-related features that drive reviewer comments. We envision the work as a study towards a review cost estimation approach. Our replication package that includes our dataset and scripts is available online.[†]

The rest of the paper is organized as follows. Section 2 presents work related to the motivation of our study. Section 3 describes our research approach. Section 4 describes our results. Section 5 presents our implications and threats to validity. Finally, Sect. 6 draws our conclusion and future work.

2. Related Work

Below we discuss related work with respect to (1) Modern Code Review, (2) the Effects of Reviewing Time, and (3) Discussions during code reviews.

Modern Code Review (MCR). The MCR is defined as a lightweight process where developers informally discuss patches in a web-based review tool such as Gerrit. The MCR is a well-established practice and has been studied in recent years. Bacchelli et al. [3] showed that reviews at Microsoft carry various benefits such as finding bugs, code improvement, knowledge transfer, team awareness, alternative solutions and shared code ownership, while Baum et al. [6] found that the web-based reviews in industry have the capability of generating better ideas. Hirao et al. [7] found that reviewers raise various discussions during the MCR process, linking reviews to others due to alternative solutions.

Various factors are related to code reviewing, and prior work has studied review quality and efficiency. Rigby et al. [8] showed that changes examined by many developers are less likely to have future defects, while Raymond [9] argued that large pools of open source reviewers are more likely to catch the most serious defects. For review efficiency, reviewer recommendation systems have been developed to automatically select appropriate reviewer candidates [10]–[14].

Effects of Reviewing Time. To evaluate how long code review processes take, recent studies have analyzed reviewing time from the first submission to final acceptance. Jiang et al. [15] found that reviewing time is impacted by technical and non-technical factors (e.g., submission time, number of modified subsystems, developer experience). Kononenko et al. [16] qualitatively analyzed how developers feel that size-related factors (e.g., patch size and number of modified files) are the most influential factors for reviewing time. Previous studies found that small patches are likely to receive faster responses [8], [17]. Baysal et al. [18] found that more experienced patch authors receive faster responses. Thongtanunam et al. [19] found that feedback delay of prior patches has a relationship with the likelihood that a patch will receive slow initial responses.

Discussions during code reviews. Recent studies have analyzed the reviewer discussion process in MCR. Gousios et al. [20] found that the number of comments is correlated with how much time the reviews take to complete. Tsay et al. [21] found that pull requests with many comments from reviewers were much less likely to be accepted. Moreover, Storey et al. [22] found that the large number of text-based messages may introduce misunderstanding through their developer survey.

To adapt the reviewers' feedback into code, patch authors revise the submitted patches and then resubmit them. Bosu et al. [23] found that patch authors feel that feedbacks from experienced reviewers can be helpful in making revisions. Rigby and Bird [4] showed that patch authors tend to resubmit roughly once to five times in six OSS projects. Tao et al. [24] qualitatively analyzed the broad reasons for patch rejection (e.g., problematic implementation, poor maintainability).

While prior work analyzed how reviewer comments impact review quality and outcome, this paper focuses on what features impact reviewer commenting.

Literature shows little research about the phenomena on commenting. To bridge this gap, we analyze factors that impact reviewer comments.

3. Methodology

Figure 1 depicts the overall methodology used in the study. It is broken into two parts, data preparation and data analysis. We describe in detail each part.

3.1 Data Preparation

Figure 1 shows how Data Preparation involves two processes of (1) system selection and (2) data cleaning.

System Selection. Table 1 shows our studied systems, which are Chromium, AOSP, Qt, Eclipse, and Libreoffice. These five systems use the Gerrit Code Review. In addition, our selected systems have been used broadly in code review studies [11], [25], [26] and span over nine years. We first collect the review history (e.g., code

[†]https://bitbucket.org/toshiki-h/commentanalysis

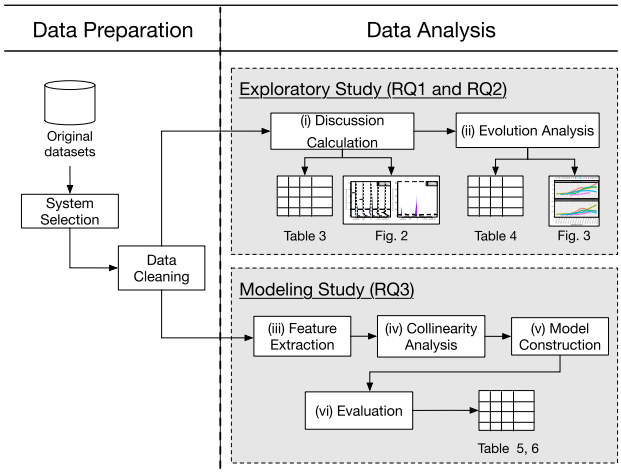


Fig. 1 Overview of our methodology. The data analysis is broken into six different stages (i-vi) and the analysis is used to answer our three research questions.

Table 1 An overview of our five studied systems.

Product	#Studied Reviews	Studied Periods	#Developers
Chromium	498,821	04/2011–10/2018	7,187
AOSP	282,203	10/2008–10/2018	6,847
Qt	207,217	07/2011–10/2018	2,446
Eclipse	116,353	04/2012–10/2018	2,143
Libreoffice	50,750	04/2012–10/2018	846
Total	1,155,344	32 years	19,469

details) through Gerrit REST API.[†] The API allows users to send queries to obtain the detailed information of patch change, reviewer and author statuses, and general and inline comments. For example, the query https://codereview.qt-project.org/changes/102938 provides users with the basic information of Qt review #102938.[††] We collect multiple types of information that we need to perform our three research questions. After collecting this information, we store those types of collected data into Mongo DB.

Data Cleaning. We clean data by removing noise in review discussion history. We are only concerned with comments that were provided by human developers. Our data clearing process comprises three steps.

Step 1. We remove comments that were generated by review bot systems. To exclude those comments, we first identify comments that include auto-generated messages (e.g, "Major sanity problems found") by review bot systems. Moreover, since review bots in our studied systems include the keyword "Bot" in their account names (e.g., Sanity Bot), we also identify comments that are given by accounts whose names include the keyword. The auto-generated messages and bot accounts that are detected by our approach are shown in the scripts of our replication package.

Step 2. We also remove build log comments that were generated by continuous integration systems. A build log comment can be identified by searching formatted messages that are generated at the beginning of its comment (e.g., "Build succeed"). Our replication package includes a list of those formatted build log messages.

Step 3. After removing those comments, we exclude reviews in which there is neither a reviewer comment nor an author comment. More specifically, after excluding comments that were generated by review bots and continuous integration systems, we count the number of general and inline comments. When there is then at least one comment, we use the review in our study. We repeat the same procedure for each studied system. Finally, to study reviews whose processes have been completed, we select reviews that were labeled "MERGED" or "ABANDONED" in our database.

3.2 Data Analysis

As shown in Fig. 1, we describe in detail the approaches we use to answer all three research questions. Those approaches are broken into six stages.

(1) Approach for RQ_1

The approach is the *(i) Discussion Calculation* stage. For this research question, we set out to understand the extent to which reviewers provide comments to complete reviews. To do so, we compute how many comments each review involves during its reviewing. In addition, to consider comment size aspect, we compute how many words are included in a reviewer comment. Moreover, in Gerrit, any developer is able to provide a general comment (i.e., placed on a reviewing board) and an inline comment (i.e., placed on a specific line of modified code). Thus, we investigate the number of general and inline comments and the number of words in a general and an inline comment separately, and analyze their distributions in our studied systems by using the `beanplot` function in the `beanplot` R package.

(2) Approach for RQ_2

The approach is the *(ii) Evolution Analysis* stage. In this analysis, we study the extent to which the number of comments and the number of words in the comments have changed over time. To conduct the analysis, we compute the average number of comments per a developer and the average number of words per a developer throughout a studied timeframe. Those two metrics can show how reviewers have changed their behaviours in commenting as studied systems evolved.

(3) Approach for RQ_3

In Fig. 1, our RQ_3 approach is split into three parts. We first describe a feature extraction to prepare studied features. Then, we describe collinearity analysis and model construction. Finally, we explain how we evaluate our models to find features that impact the number of comments and number of words in the comments per a review. We explain in detail each stage.

The *(iii) Feature Extraction* stage studies the overall

[†]https://gerrit-review.googlesource.com/Documentation/rest-api.html

[††]https://codereview.qt-project.org/c/qt-creator/qt-creator/+/102938

perspectives in code review activities. We choose features based on prior work [19]. Table 2 describes each feature with its definition. In summary, our selected features are divided into three segments. The first segment is *Patch Property* in which its features characterize patch information. The second segment is *Human Experience* that provides features related to the activity of developers who have submitted or reviewed patches in the past. The third segment is *Project Management* where features quantify the extent to which the workload is intense in a project.

The *(iv) Collinearity Analysis* stage identifies and removes highly correlated features and redundant features due to the risk for disturbing results of our modeling analysis. Highly correlated features are features that have a high correlation with other ones. However, after removing highly correlated features, some features might still quantify the same phenomenon and show homogeneous outcomes. Those features are considered as the redundant features. To counteract those two types of features, we conduct two methods to eliminate high correlation and redundancy between our features. We measure the possible correlation among our features using Spearman's rank correlation coefficients (ρ) to remove highly correlated features. Similar to prior work [19], we use Spearman's rank correlation coefficient $|\rho| = 0.7$ as the threshold to eliminate highly correlated features. Furthermore, features that do not have a high correlation might still be redundant and result in misleading conclusions [27]. To remove them, we use the `redun` function in the `rms` R package. The `redun` function uses a flexible additive model to predict each feature from remaining features to find the redundant feature(s).[†]

The *(v) Model Construction* stage is when we train regression models using our features. A regression model is a commonly used type that describes the relationship between a dependent variable and independent variables. From our preliminary investigation, we decided to use a logarithmic scale in independent variables for the best performance. To construct models, we use the `lm` function in the R package. Moreover, since our features are computed in their different approaches, we standardize the coefficients that our regression models generate to measure the degree of impact for each feature. To compute standardized coefficients, we use the `lm.beta` function in the `lm.beta` R package.

Finally, the *(vi) Evaluation* stage analyzes the importance of each feature for general and inline comments. The evaluation is done by computation of the standardized coefficient of each feature that explains the degree to which the feature impacts the number of comments and number of words in the comments. We then analyze each high value of the coefficients and use them as discussions for rationale.

Table 2 The description of our selected features in Patch Property, Human Experience and Project Management segments.

Feature	Description
Patch Property	
Patch Churn	Number of lines added to and deleted from changed files.
#Subsystems	Number of subsystems that are changed.
#Directories	Number of directories that are changed under the subsystem(s). For example, in QT review #265820, since the modification appears in three different directories (coreplugin, projectexplorer, vcsbase) under one subsystem (qt-creator), #subsystems and #directories are counted as one and three, respectively.
Description Length	The length of a commit message i.e., Number of words.
Purpose (Doc)	Whether the purpose of a patch is documentation.
Purpose (Feature)	Whether the purpose of a patch is feature introduction.
Human Experience	
Author Experience	Number of reviews that an author has submitted in past development. For example, suppose an author has made three submissions under subsystem A and two submissions under subsystem B in a past development, we count the author's experience as five (submissions).
Reviewers Experience	Number of reviews that reviewers have reviewed in past development. For example, suppose a review involves two reviewers one of which has reviewed five past submissions and another has reviewed ten past submissions, we count the reviewers' experience as 15 (reviews).
Project Management	
Overall Workload	Number of reviews that have been submitted in a certain period (i.e., within 7 days).
Directory Workload	Number of reviews that have been submitted under the same directory in a certain period (i.e., within 7 days).
#Days since Last Modification	Number of days since the last time when a file in a patch was modified
#Prior Defects	Number of prior defects that have appeared in a file of a patch under review.

4. Results

In this section, we provide the results to each of the research questions. First, we highlight our findings and then answer each question.

4.1 Answering RQ_1

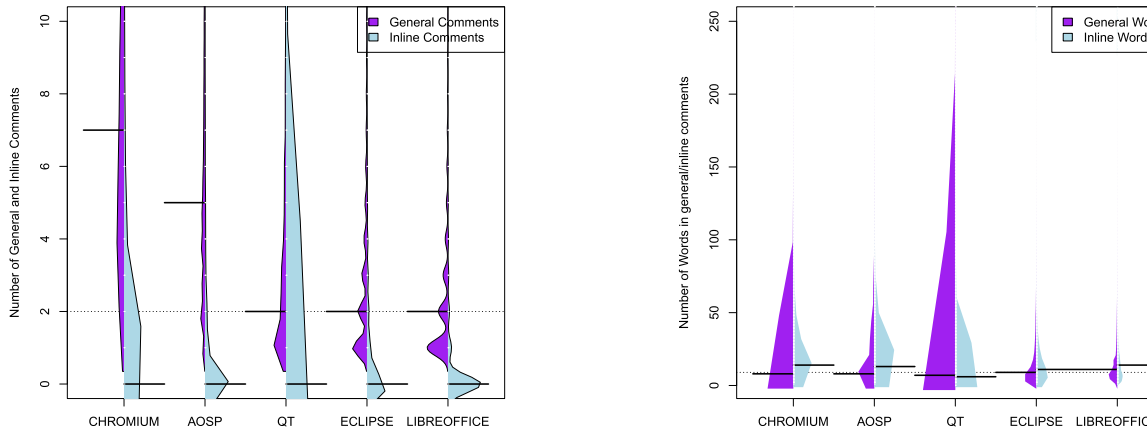
Table 3 and Fig. 2 show the results of RQ_1. We were able to make two observations as outlined below.

The number of comments and number of words in a comment vary among reviews and across systems. Ta-

Table 3 The statistics of numbers of total comments, general comments and inline comments for each system.

Project	Min.	1st Qu.	Med.	Mean	3rd Qu.	Max.
Total Comments (i.e., the sum of general and inline comments)						
Chromium	1	4	8	12.59	14	1,372
AOSP	1	3	6	9.27	11	503
Qt	1	1	3	6.68	6	2,632
Eclipse	1	1	3	5.23	5	338
Libreoffice	1	1	2	2.99	3	81
General Comments						
Chromium	1	4	7	10.49	13	905
AOSP	1	3	5	7.86	10	247
Qt	1	1	2	4.06	5	367
Eclipse	1	1	2	3.31	4	106
Libreoffice	1	1	2	2.37	3	34
Inline Comments						
Chromium	0	0	0	2.10	1	1,144
AOSP	0	0	0	1.41	1	365
Qt	0	0	0	2.62	1	2,630
Eclipse	0	0	0	1.92	1	232
Libreoffice	0	0	0	0.62	0	70

[†]https://www.rdocumentation.org/packages/Hmisc/versions/4.2-0/topics/redun



(a) Number of comments per a review

(b) Number of words per a comment

Fig. 2 The distributions of the number of general and inline comments and number of words in the comments across our studied systems.

ble 3 shows that the number of total comments approximate 9–13 on average in the two largest studied systems (i.e., Chromium and AOSP). In contrast, Qt, Eclipse, and Libreoffice reviews comprise 3–7 comments on average. We assume that the results are reflective of a system size (i.e., the number of reviews). Indeed, we observe that the larger a system size is, the more comments the system's reviews garner (see Table 1 and Table 3). Furthermore, inline comments have the tendency to include more words than general comments. Figure 2 (a) and Fig. 2 (b) show that despite general comments are provided more frequently than inline comments, inline comments tend to involve more words than general comments. For example, Fig. 2 (b) shows that the median of the number of words in a inline comment is larger than the number of words in a general comment in Chromium, AOSP, Eclipse, and Libreoffice.

Prior studies found that useful comments are considered as comments that trigger code changes [23], and are different from non-useful comments with regard to several textual properties [28]. Basically, inline comments are provided to trigger specific code changes, requiring much context to clarify such needs. Our results complement prior studies, suggesting that inline comments have the potential to include more insights than general comments in terms of comment size.

Reviewers treat general and inline comments differently. Table 3 and Fig. 2 (a) show that general comments often appear during reviewing at 3–13 comments on average, whereas reviewers rarely provide inline comments. We suspect that developers tend to discuss the abstract impact of the patch rather than looking into specific modification or enhancement of the submitted patch. Interestingly, the distributions in Fig. 2 (a) visually show that developers tend to write more general comments than inline comments. Indeed, this figure shows that a majority of the Libreoffice reviews contained no inline comments (i.e., the inline count is zero). Table 3 shows that 25% of reviews (above 3rd Qu.) in the two most studied and largest systems (i.e., Chromium and AOSP) used more than ten general comments to make their final decisions. Although MCR is known as lightweight [3], those systems have the potential for requiring more comments.

Table 4 The correlations between the number of general and inline comments and the number of words in general and inline comments.

Comment Type	Chromium	AOSP	Qt	Eclipse	Libreoffice
General	0.23	0.65	0.30	0.74	0.76
Inline	0.89	0.83	0.67	0.85	0.77

Table 4 shows that the number of inline comments per a review have a substantial correlation (0.67–0.89) with the number of words in the inline comments per a review across our studied projects. This result indicates that the number of words in inline comments is mostly consistent in each studied system, proportionally increasing with the number of inline comments. Indeed, we find that AOSP, Eclipse, and Libreoffice systems have a substantial correlation (0.65–0.74) between the number of general comments and the number of words in the general comments, while Chromium and Qt systems do not. General comments are provided not only to discuss the abstract impacts, but also to simply show their agreements, which may cause the inconsistent degree of correlation in general comments among studied systems.

We return to answer *(RQ_1) Is there a typical number of reviewer comments before the final decisions?*:

> *There is no typical number of review comments across five studied systems. Reviewers reach a decision on a review ranging up to 13 comments, with 22 words per a comment on average. Moreover, the number of comments is mostly correlated with the number of words in the comments.*

4.2 Answering RQ_2

Figure 3 shows the results of RQ_2. From this figure, we make two following observations:

Reviewer comments have increased over time for Chromium. Figure 3 shows that the average number of total comments that a reviewer provides in a review has increased over time in the largest system Chromium. For example, the system reached the first peak in late 2014. After this peak, its average number of total comments per a developer grew rapidly again in recent months. Conversely, the number of words per a developer has decreased after the peak in Chromium, implying that reviewers are likely to participate more actively rather than just commenting. Our results complement prior work [4], suggesting that the number of comments in modern code review environments has the potential to increase as a system evolves.

In contrast, reviewer comments tend to stabilize or stall in other studied systems. Figure 3 shows that the average number of total comments per a developer in AOSP and Qt systems has increased until early 2016, while the average number roughly tended to stabilize in recent months. Although the number of total comments per a developer in Eclipse and Libreoffice has the slight increase from their initial start until mid 2016, those numbers stalled in recent months. We assume that the project which has a frequent release schedule may induce more comments because developers feel pressure to catch up with every release. Indeed, the Chromium system has a major release every month, while the rest of our studied systems release a main version once every six months or twelve months. Moreover, Fig. 3 also shows that the number of words per a developer has a stable tendency to the number of comments per a developer

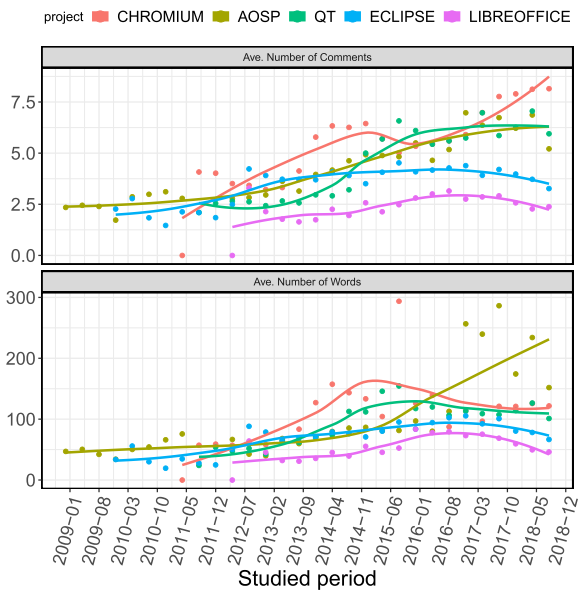


Fig. 3 The evolution of the (i) number of total comments per a developer and (ii) number of total words per a developer throughout a studied timeframe.

in AOSP, Eclipse, and Libreoffice over time. For example, the number of comments per a developer and number of words per a developer in Libreoffice gradually increased until late 2016. After that time, the decrease tendency has been found in recent periods. We observe that an increase in correlations between comment and word aspects (see Table 4) across AOSP, Eclipse, and Libreoffice had led to the similar tendencies of the evolution between the number of comments per a developer and the number of words per a developer. However, Fig. 3 shows that the number of words per a developer in Chromium and Qt has decreased, while the number of comments per a developer in those systems has increased rapidly or gradually over time. This inverse result might be related to the weak correlations of those two systems that we showed in the second observation of the previous research question.

We now return to answer *(RQ_2) Does reviewer commenting change over the evolution of a project?*:

> *The number of comments tends to steadily increase over time in the largest studied system. Furthermore, the number of comments tends to mostly stabilize in other studied systems.*

4.3 Answering RQ_3

Table 5 and Table 6 show our twelve features that remain after collinearity and redundancy analyses. To answer RQ_3, we make two observations.

Human experience features can drive general comments across studied systems. Table 5 shows that for both the number of general comments and number of words in the general comments, reviewer experience is the most impactful feature in all studied systems. Specifically, reviewer experience has an increase impact across our studied systems, indicating that experienced reviewers tend to provide more general comments and include more words. This is because experienced reviewers are capable of pointing out broader issues than inexperienced ones. In contrast, author experience shows a decrease impact across our studied systems, suggesting that since experienced authors are more knowledgeable than novice ones, they can address issues that may induce discussion before they submit patches. We find that the more the author is experienced, the less likely that the number of general comments and the number of words in the general comments will decrease across our studied systems.

Table 5 also shows that patch churn can be observed as a relatively impactful feature in Chromium, AOSP, Qt, and Eclipse for the number of general comments and number of words in the general comments. This result suggests that the larger the patch churn is, the more likely the review will receive comments and words. Moreover, dispersion of changes (i.e., #Subsystems and #Directories) mostly shows the increase impact on the number of general comments and number of words in the general comments. The results im-

Table 5 The standardized coefficient of each feature for general comments and words in the general comments. The blue colour cell depicts the most impactful metric.

Feature	General Comments					General Words				
	Chromium	AOSP	Qt	Eclipse	Libreoffice	Chromium	AOSP	Qt	Eclipse	Libreoffice
Patch Property										
Patch Churn	0.16	0.13	0.15	0.12	0.05	0.10	0.06	0.11	0.12	0.01
#Subsystems	0.04	–	0.14	0.01	0.04	0.03	–	0.18	0.01	0.04
#Directories	0.11	–	-0.04	–	0.01	0.02	–	-0.05	-	0.01
Description Length	0.02	0.07	0.10	0.12	0.03	0.06	0.08	0.08	0.14	0.05
Purpose (Doc)	0.02	0.01	-0.01	< 0.01	0.02	< 0.01	< 0.01	-0.02	< 0.01	0.02
Purpose (Feature)	0.03	0.03	0.02	0.01	< 0.01	0.02	0.01	< 0.01	-0.01	-0.01
Human Experience										
Author Experience	-0.10	-0.04	-0.13	-0.10	-0.21	-0.08	0.01	-0.23	-0.14	-0.23
Reviewers Experience	0.34	0.41	0.19	0.28	0.23	0.68	0.75	0.24	0.36	0.27
Project Management										
Overall Workload	0.10	0.16	0.02	< 0.01	-0.02	0.02	0.04	0.05	-0.07	-0.03
Directory Workload	0.04	< 0.01	0.01	< 0.01	0.01	0.07	-0.01	0.04	-0.01	< 0.01
#Days since Last Modification	0.03	0.02	0.01	< 0.01	0.03	0.04	0.03	< 0.01	< 0.01	0.01
#Prior Defects	-0.03	0.03	0.05	< 0.01	0.02	-0.07	0.01	0.02	-0.01	0.01

Table 6 The standardized coefficient of each feature for inline comments and words in the inline comments. The blue colour cell depicts the most impactful metric. Note that the standardized coefficient of Patch Churn (0.145) is larger than of Reviewers Experience (0.138) in Eclipse.

Feature	Inline Comments					Inline Words				
	Chromium	AOSP	Qt	Eclipse	Libreoffice	Chromium	AOSP	Qt	Eclipse	Libreoffice
Patch Property										
Patch Churn	0.22	0.19	0.10	0.14	0.10	0.30	0.22	0.29	0.16	0.10
#Subsystems	0.04	–	< 0.01	-0.01	0.02	0.05	–	0.02	-0.02	0.03
#Directories	-0.02	–	0.03	–	-0.01	-0.04	–	-0.04	–	-0.01
Description Length	0.03	0.04	-0.01	0.05	-0.02	0.03	0.07	0.11	0.08	-0.03
Purpose (Doc)	0.03	0.01	< 0.01	0.01	< 0.01	0.02	0.01	0.01	< 0.01	< 0.01
Purpose (Feature)	0.03	0.02	< 0.01	0.02	0.01	0.04	0.02	0.02	0.02	0.01
Human Experience										
Author Experience	-0.12	-0.05	-0.01	-0.08	-0.14	-0.15	-0.06	-0.13	-0.10	-0.19
Reviewers Experience	0.10	0.01	0.02	0.14	0.08	0.24	0.11	0.21	0.30	0.17
Project Management										
Overall Workload	< 0.01	0.04	< 0.01	0.02	0.01	-0.01	0.01	-0.01	0.02	0.04
Directory Workload	0.01	< 0.01	-0.01	0.01	0.02	0.05	0.03	< 0.01	0.02	0.03
#Days since Last Modification	0.02	< 0.01	< 0.01	0.01	0.04	0.08	0.04	0.03	0.03	0.06
#Prior Defects	-0.01	0.03	< 0.01	0.01	0.02	-0.07	0.03	0.07	< 0.01	0.05

ply that when modification spreads across multiple subsystems or directories, reviewers may raise a broader discussion to avoid unexpected problems that might affect external components. Indeed, maintainability issues are raised more frequently than functional defects in reviews [29], [30].

The project management segment shows both increase and decrease impacts across our studied systems. For example, overall workload feature is increased and relatively impactful compared to other features in AOSP. The results imply that the impact of the project management segment varies due to a system size. Indeed, our RQ2 shows that the growth of the number of comments and number of words in the comments vary across systems (see Fig. 3).

Patch property features are likely to drive inline comments. However, different from the number of general comments and number of words in the general comments, patch churn feature plays a considerable role on both the number of inline comments and number of words in the inline comments. Table 6 shows that Patch Churn is ranked at the 1st place in Chromium, AOSP, Qt, and Eclipse. Similarly, the feature is most considerable in Chromium, AOSP, and Qt for the number of words in the inline comments. This result indicates that if the modified lines of code increase, the number of inline comments and number of words in the inline comments will rise due to various issues (e.g., typo, code style).

Table 6 also shows that author experience has a decrease impact on the number of inline comments and words in the inline comments, similar to general comments. Moreover, reviewer experience has an increase impact on the number of inline comments and the number of words in the comments. Similar to our previous observation, our results suggest that novice authors and experienced reviewers can increase the number of inline comments and number of words in the inline comments. Besides, both increase and decrease impacts on the number of inline comments and number of words in the inline comments are shown in overall workload and directory workload features across our studied systems, implying that the impact of project management features for inline comments depends on a system's size.

We now answer *(RQ_3) What (a) patch, (b) human and (c) management features drive reviewer comments?*:

> Human experience and patch property features drive reviewer comments and words in the comments. Moreover, both novice authors and experienced reviewers tend to induce more comments and words, while the large and widespread modifications also have the tendency to raise more comments and words across studied systems.

5. Discussion

In this section, we discuss the implications and the threats to the validity of this study.

5.1 Implications

Based on the results, we now discuss some of the implications and actionable contributions from our findings.

1. **There are no magic number of comments and number of words to complete reviews.** Our results suggest that reviewers can complete a review in 3–13 comments on average. This is a wide range, showing that review participation can range and cannot be fit into a smaller range. Importantly, it is possible to identify review participation (number of comments) and the comment size (number of words in a comment) that go outside of their ranges as either underwhelming or becoming bloated. This can become actionable, as we can now flag and identify these abnormal reviews outside of their ranges.
2. **The number of comments and number of words fluctuate over time.** Our results show that the number of comments and number of words in the comments change over the lifetime of the system. Moreover, we also find that the number of comments and the number of words in the comments have the potential to increase as studied systems grow. Due to a lazy consensus i.e., reviewers responding only to necessary contents, a lack of interest or maybe if there is an overload of developers, there is no incentive to respond to every comment. For future work, we would like to also explore how commenting is affected by the external factors, such as being a newcomer to a project, the increase in source code and review requests, and the maturity of the software project. The actionable implication is that we now know that outside factors have an impact on reviewer commenting.
3. **Human experience and patch property features have the strongest impact on general and inline comments, respectively.** In RQ_3, our results show that novice authors or experienced reviewers can increase the number of general comments and number of words in the general comments across our studied systems. Our results also show that patch churn and dispersion of modified codes can increase the number of inline comments and number of words in the inline comments. We suggest that to minimize the discussion, the experience of the author and reviewers and the property size of patches should be more carefully considered before starting with the discussion.

We envision this paper as working towards a review cost estimation approach. In our future plans, we would like to (i) analyze the relationship between review cost and quality; and (ii) build models that automatically determine the extent to which a review will take a certain time by machine learning techniques. We envision that this work can also open up research into the effective management of code reviews.

5.2 Threats to Validity

In this section, we present construct, internal and external threats to our study.

(1) Construct Validity

Construct threats to validity refer to the concerns between the theory and achieved results of the study. To prevent noise from interfering with our observations, we clean our raw data by removing potential noisy comments. However, there might be cases where our comment extraction mistakenly excludes actual human comments or includes comments of CI or Bot systems. Thus, we preliminarily investigated continuous integration log and bot comments, and then excluded comments that were generated by those automated tools.

(2) Internal Validity

Internal threats to validity refer to the concerns that are internal to the study. We discuss two threats. The first threat is the coverage of the features used in the study. We analyze a broad range of patch property, human experience, and project management features; nonetheless, our study has not covered other possible aspects. Therefore, to mitigate the threat, we select features based on similar prior work [19], [25], [31].

The second threat is the validity of our methodology for the modeling study. There might be cases where we could use another sophisticated algorithm. However, we choose regression models which prior studies have used for years [19], [32].

(3) External Validity

External threats to validity refer to the generalization concerns of this study's results. The main external threat is the generality of our studied systems. Although we collected studied systems based on their popularity and quality of their datasets, there might be a threat such that our results do not generalize to all software systems. We chose five studied systems in terms of their popularity in the code review research field to ensure that our studied systems show valid information. Therefore, we are confident that we have addressed the threads of our studied systems as they have been used in prior studies [11], [32]–[34] and were released as official MSR datasets [26], [35].

6. Conclusions

We conducted exploratory and modeling studies over 1.1 million reviews across five studied systems (i.e., Chromium, AOSP, Qt, Eclipse, and Libreoffice). Through those studies, our results indicate that:

- The number of both general and inline comments varies among reviews and across studied systems.
- Reviewers change their behaviours in commenting as a system evolves.
- Reviewer comments are most likely to be affected by developer experience and patch property size.

The main goal of this work is to move towards a review cost estimation approach for effective code reviews in the modern code review. We also hope that this research has implications for future research.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Number 17J09333, 17H00731, and 18H04094, and the Support Center for Advanced Telecommunications (SCAT) Technology Research, Foundation.

References

[1] K.E. Wiegers, Peer Reviews in Software: A Practical Guide, Addison-Wesley Longman Publishing Co., Inc., 2002.
[2] M.E. Fagan, "Design and code inspections to reduce errors in program development," IBM Systems Journal, vol.15, no.3, pp.182–211, 1976.
[3] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," Proc. 35th International Conference on Software Engineering, pp.712–721, 2013.
[4] P.C. Rigby and C. Bird, "Convergent contemporary software peer review practices," Proc. Joint Meeting on Foundations of Software Engineering, pp.202–212, 2013.
[5] J. Czerwonka, M. Greiler, and J. Tilford, "Code reviews do not find bugs: How the current code review best practice slows us down," Proc. 37th International Conference on Software Engineering - Volume 2, ICSE '15, Piscataway, NJ, USA, pp.27–28, IEEE Press, 2015.
[6] T. Baum, O. Liskin, K. Niklas, and K. Schneider, "Factors influencing code review processes in industry," Proc. 24th International Symposium on Foundations of Software Engineering, pp.85–96, Nov. 2016.
[7] T. Hirao, S. McIntosh, A. Ihara, and K. Matsumoto, "The Review Linkage Graph for Code Review Analytics: A Recovery Approach and Empirical Study," Proc. International Symposium on the Foundations of Software Engineering, pp.578–589, 2019.
[8] P.C. Rigby, D.M. German, and M.-A. Storey, "Open source software peer review practices: a case study of the apache server," Proc. 30th International Conference on Software Engineering, pp.541–550, 2008.
[9] E.S. Raymond, "The cathedral and the bazaar," Knowledge, Technology & Policy, vol.12, no.3, pp.23–49, 1999.
[10] V. Balachandran, "Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation," Proc. 35th International Conference on Software Engineering, pp.931–940, 2013.
[11] P. Thongtanunam, C. Tantithamthavorn, R.G. Kula, N. Yoshida, H. Iida, and K. Matsumoto, "Who should review my code? a file location-based code-reviewer recommendation approach for modern code review," Proc. 22nd International Conference on Software Analysis, Evolution, and Reengineering, pp.141–150, 2015.
[12] X. Xia, D. Lo, X. Wang, and X. Yang, "Who should review this change? putting text and file location analyses together for more accurate recommendations," Proc. 31st International Conference on Software Maintenance and Evolution, pp.261–270, 2015.
[13] M.B. Zanjani, H. Kagdi, and C. Bird, "Automatically recommending peer reviewers in modern code review," IEEE Trans. Softw. Eng., vol.42, no.6, pp.530–543, June 2016.
[14] A. Ouni, R.G. Kula, and K. Inoue, "Search-based peer reviewers recommendation in modern code review," 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp.367–377, Oct. 2016.
[15] Y. Jiang, B. Adams, and D.M. German, "Will my patch make it? and how fast?: Case study on the linux kernel," Proc. 10th Working Conference on Mining Software Repositories, pp.101–110, 2013.
[16] O. Kononenko, O. Baysal, and M.W. Godfrey, "Code review quality: How developers see it," Proc. 38th International Conference on Software Engineering (ICSE), pp.1028–1038, 2016.
[17] P. Weißgerber, D. Neu, and S. Diehl, "Small patches get in!," Proc. International Working Conference on Mining Software Repositories (MSR'08), pp.67–76, 2008.
[18] O. Baysal, O. Kononenko, R. Holmes, and M.W. Godfrey, "The influence of non-technical factors on code review," Proc. 20th Working Conference on Reverse Engineering, pp.122–131, 2013.
[19] P. Thongtanunam, S. McIntosh, A.E. Hassan, and H. Iida, "Review participation in modern code review: An empirical study of the android, qt, and openstack projects," Empirical Software Engineering, vol.22, no.2, p.768–817, 2017.
[20] G. Gousios, M. Pinzger, and A.v. Deursen, "An exploratory study of the pull-based software development model," Proc. 36th International Conference on Software Engineering, ICSE 2014, pp.345–355, 2014.
[21] J. Tsay, L. Dabbish, and J. Herbsleb, "Influence of social and technical factors for evaluating contribution in github," Proc. 36th International Conference on Software Engineering, ICSE'14, pp.356–366, 2014.
[22] M.-A. Storey, A. Zagalsky, F.F. Filho, L. Singer, and D.M. German, "How social and communication channels shape and challenge a participatory culture in software development," IEEE Trans. Softw. Eng., vol.43, no.2, pp.185–204, Feb. 2017.
[23] A. Bosu, M. Greiler, and C. Bird, "Characteristics of useful code reviews: An empirical study at microsoft," Proc. 12th International Working Conference on Mining Software Repositories, pp.146–156, 2015.
[24] Y. Tao, D. Han, and S. Kim, "Writing acceptable patches: An empirical study of open source project patches," Proc. International Conference on Software Maintenance and Evolution, pp.271–280, 2014.
[25] T. Fukushima, Y. Kamei, S. McIntosh, K. Yamashita, and N. Ubayashi, "An empirical study of just-in-time defect prediction using cross-project models," Proc. 11th Working Conference on Mining Software Repositories, MSR 2014, pp.172–181, 2014.
[26] X. Yang, R.G. Kula, N. Yoshida, and H. Iida, "Mining the modern code review repositories: A dataset of people, process and product," Proc. 13th Working Conference on Mining Software Repositories, pp.460–463, 2016.
[27] J. Jiarpakdee, C. Tantithamthavorn, A. Ihara, and K. Matsumoto, "A study of redundant metrics in defect prediction datasets," Proc. International Symposium on Software Reliability Engineering, pp.51–52, 2016.
[28] M.M. Rahman, C.K. Roy, and R.G. Kula, "Predicting usefulness of code review comments using textual features and developer experience," Proc. 14th International Conference on Mining Software Repositories, pp.215–226, 2017.
[29] M.V. Mäntylä and C. Lassenius, "What types of defects are really discovered in code reviews?," IEEE Trans. Softw. Eng., vol.35, no.3, pp.430–448, 2009.
[30] M. Beller, A. Bacchelli, A. Zaidman, and E. Juergens, "Modern code reviews in open-source projects: Which problems do they fix?," Proc. Working Conference on Mining Software Repositories, pp.202–211, 2014.
[31] Y. Kamei, E. Shihab, B. Adams, A.E. Hassan, A. Mockus, A. Sinha, and N. Ubayashi, "A large-scale empirical study of just-in-time qual-

ity assurance," IEEE Trans. Softw. Eng., vol.39, no.6, pp.757–773, 2013.
[32] S. McIntosh, Y. Kamei, B. Adams, and A.E. Hassan, "An empirical study of the impact of modern code review practices on software quality," Empirical Software Engineering, vol.21, no.5, pp.2146–2189, 2016.
[33] P. Thongtanunam, S. McIntosh, A.E. Hassan, and H. Iida, "Revisiting code ownership and its relationship with software quality in the scope of modern code review," Proc. 38th International Conference on Software Engineering, pp.1039–1050, 2016.
[34] S. McIntosh, Y. Kamei, B. Adams, and A.E. Hassan, "The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects," Proc. 11th Working Conference on Mining Software Repositories, pp.192–201, 2014.
[35] K. Hamasaki, R.G. Kula, N. Yoshida, A.E.C. Cruz, K. Fujiwara, and H. Iida, "Who does what during a code review? datasets of oss peer review repositories," Proc. Working Conference on Mining Software Repositories, pp.49–52, 2013.

Toshiki Hirao is a PhD student at Nara Institute of Science and Technology, Japan. He has been a doctoral course fellowship student (DC1) in JSPS from 2017 to present. His PhD thesis aims to improve code review efficiency. He received Master's degree from Nara Institute of Science and Technology, Japan.

Raula Gaikovina Kula is an assistant professor at Nara Institute of Science and Technology. He received the Ph.D degree from Nara Institute of Science and Technology in 2013. His interests include Software Libraries, Software Ecosystems, Code Reviews and Mining Software Repositories.

Akinori Ihara is a lecturer at Wakayama University in Japan. His research interests include empirical software engineering, open source software engineering, social software engineering and mining software repositories (MSR). His work has been published at premier venues like ICSE, MSR, and ISSRE. He received the M.E. degree (2009) and Ph.D. degree (2012) from Nara Institute of Science and Technology.

Kenichi Matsumoto is a professor in the Graduate School of Science and Technology at Nara Institute of Science and Technology, Japan. He received the Ph.D. degree in Engineering from Osaka University. His research interests include software measurement and the software process. He is a fellow of the IEICE and the IPSJ, a senior member of the IEEE, and a member of the JSSST.