How Does Code Reviewing Feedback Evolve?

A Longitudinal Study at Dell EMC

Ruiyin Wen*
McGill University, Canada
ruiyin.wen@mail.mcgill.ca

Maxime Lamothe
Polytechnique Montréal, Canada
maxime.lamothe@polymtl.ca

Shane McIntosh
University of Waterloo, Canada
shane.mcintosh@uwaterloo.ca

ABSTRACT

Code review is an integral part of modern software development, where fellow developers critique the content, premise, and structure of code changes. Organizations like Dell EMC have made considerable investment in code reviews, yet tracking the characteristics of feedback that code reviews provide (a primary product of the code reviewing process) is still a difficult process. To understand community and personal feedback trends, we perform a longitudinal study of 39,249 reviews that contain 248,695 review comments from a proprietary project that is developed by Dell EMC. To investigate generalizability, we replicate our study on the OpenStack Nova project. Through an analysis guided by topic models, we observe that more context-specific, technical feedback is introduced as the studied projects and communities age and as the reviewers within those communities accrue experience. This suggests that communities are reaping a larger return on investment in code review as they grow accustomed to the practice and as reviewers hone their skills. The code review trends uncovered by our models present opportunities for enterprises to monitor reviewing tendencies and improve knowledge transfer and reviewer skills.

ACM Reference Format:
Ruiyin Wen, Maxime Lamothe, and Shane McIntosh. 2022. How Does Code Reviewing Feedback Evolve?: A Longitudinal Study at Dell EMC. In *44th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '22), May 21–29, 2022, Pittsburgh, PA, USA*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3510457.3513039

1 INTRODUCTION

Code review is a process whereby fellow developers inspect code changes and provide feedback. It is a mechanism by which software teams improve software quality [32], signal project progress [2], and collaboratively solve problems [39]. Nowadays, teams adopt tools to help coordinate the code review process, which provide an online interface and store data in a code review database.

Unlike the rigid code inspections of the past [14], the modern variant of the code review process is informal; however, review discussions are still a rich source of information about the evolution of the system under review. Bacchelli and Bird [2] found that motivations for code review are both technical (e.g., catching defects early) and non-technical (e.g., promote knowledge transfer). Rigby and Bird [39] found that the focus of code review has shifted from defect hunting to collaborative problem solving. Indeed, recent work has reported that roughly 75% of the issues that are uncovered [28] and fixed [6] during code review do not alter system behaviour, instead aiming to improve system maintainability.

The value produced by a code review process is dependent on the investment that reviewers make. For example, the existence of a code review has been shown to share a weaker link with quality indicators than the measures of review participation [31].

Little is known about how reviewing feedback (a primary value-generating artifact of the code review process) changes as a community and its stakeholders mature. Teams at Dell EMC have invested in code review for several years, yet management lacked the appropriate instruments and tools to understand how reviewing feedback has been evolving. In particular, management at Dell EMC would benefit from a cost-effective way to track community and personal development trends.

To that end, in this paper, we present an empirical study of code review feedback at Dell EMC. We also replicate our approach on the OpenStack community to compare our observations and generate a dataset for reproducibility.[1] More specifically, we train and analyze topic models using a corpus of 248,695 comments from 39,249 code change reviews (Section 2). These models show that context-specific issues (e.g., API-related topics) are more prevalent than formatting issues (Section 3). We then use the models to perform longitudinal studies (Sections 4 & 5), which address the following questions:

RQ1. How does the prevalence of code review topics change as a community ages?
Motivation: Prior work [41, 47] has analyzed the content of code review discussions; however, little is known about how this changes as a community matures. For example, after introspection, a community may adjust its reviewing focus to address its perceived shortcomings. Alternatively, a community may tacitly degrade in its code reviewing focus. Thus, we first set out to explore how the content of review discussions change as a software community matures. For this RQ, we concentrate on the overall maturity (age) of the software community as an organization and set aside the experience of developers in the community.
Results: Reviewing behaviour is rapidly evolving in Dell EMC. Our topic models track these shifts in community focus. For example, our results show that context-specific feedback has become more prevalent over time. We also observe shifting trends in *code review process*, *logging and*

*The bulk of this work was completed during Mr. Wen's internship at Dell EMC.

ICSE-SEIP '22, May 21–29, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9226-6/22/05…$15.00
https://doi.org/10.1145/3510457.3513039

[1] https://figshare.com/s/d227f9f489898ab051bc

Table 1: An overview of the studied projects.

Project	Scope	#Changes	#Cmts	Years
Dell EMC	Enterprise data backup & recovery	12,702	94,524	4
OpenStack Nova	Provisioning management for OpenStack	26,547	154,171	6
Total	-	39,249	248,695	10

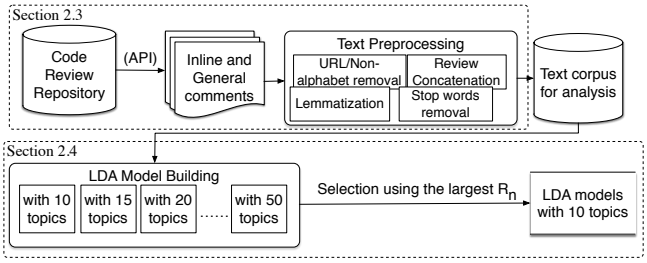

Figure 1: The data preparation and model training processes.

error messages, and *object-oriented design* in a period that coincides with large changes in team composition.

RQ2. How does the prevalence of code review topics change as reviewers accrue experience?

Motivation: A software community is made up of developers who are ideally growing and improving. Prior work [9, 23, 40] has shown that the more experienced reviewers are often the authors of review feedback that is perceived to be of higher quality. Hence, we set out to better understand how reviewers change their focus as they accrue experience.

Results: Coarse-grained experience indicators like developer promotion to core teams do not coincide with a significant change in reviewing topics; however, reviewing behaviour does evolve when finer-grained experience indicators are considered. Our topic models show that experienced reviewers specialize in different ways according to the needs of their communities. The more experienced Dell EMC reviewers tend to focus more on context-specific, technical feedback, suggesting that their reviewing skills are honed to provide feedback with a greater return on investment. In addition, we observe trends that coincide with team focus. For example, as Dell EMC reviewers accrue experience, they tend to provide more *code style* feedback, which coincides with a mentorship investment that senior Dell EMC developers have made to help in onboarding a recent influx of new developers.

2 CASE STUDY DESIGN

Below, we describe the subject projects (Section 2.1), summarize the code review process (Section 2.2), and explain our approaches to data extraction (Section 2.3) and training topic models (Section 2.4).

2.1 Studied Projects

The primary studied data set is extracted a proprietary project developed by Dell EMC. The project is large and rapidly evolving with a globally distributed development community and userbase. Table 1 provides an overview of the studied projects. The Dell EMC project provides an enterprise solution that orchestrates data backup for disaster recovery. Customers of the Dell EMC solution include major players in several market sectors, such as financial, aerospace, and educational institutions.

Due to legal constraints, we cannot share all of the technical details of our analysis of Dell EMC. Therefore, we complement our analysis of Dell EMC with an open source project, OpenStack Nova, which allows us to share a replication package[1] and improve the reproducibility of our paper. The studied OpenStack community develops software that manages large pools of compute, storage, and networking resources, and is used to support a wide array of business applications. We analyze Nova, the provisioning management system, because it attracts the most developers when compared to the other OpenStack projects. As a cross-company open source community, OpenStack has a vested interest in improving their code review process. In the past, researchers have used data from the OpenStack community to evaluate reviewer recommendation approaches [45], analyze the relationship between reviewing and authoring expertise [44], evaluate the fairness of code reviews [15], study the career paths of developers [48], and study company participation in open source development [16].

2.2 The Code Review Process

We analyze review comments from the Dell EMC and OpenStack Nova code review repositories. To provide context to our analysis, we describe the code review process of OpenStack Nova. A similar process is employed at Dell EMC.

The OpenStack community uses Gerrit—a web-based review tool that tightly integrates with the Git version control system. The code review process is comprised of five steps:

(1) **The author uploads changes to the Gerrit server.** A review is initiated by uploading changes to Gerrit.
(2) **The Gerrit server performs automatic verification.** As a part of Continuous Integration (CI), the Gerrit server of the OpenStack community initiates verification of the change to check for simple coding mistakes (e.g., through linters and automated testing).
(3) **The reviewers inspect the changes and initiate discussion.** Reviewers may provide *inline comments*, i.e., comments that correspond to lines within the change.
(4) **The author replies to the reviewers and/or revises her code (if necessary).** The author may discuss with reviewers by replying to their comments. If the change is not approved for integration (e.g., due to insufficient support from reviewers or verification failure), the author may improve the change by addressing the concerns.
(5) **Integrate the approved code.** Steps 1–4 may be repeated multiple times. Hence, a change may incur multiple rounds of code review. Once the reviewers are satisfied with the code, the author may add the change to the queue for integration.

2.3 Data Extraction and Preprocessing

Figure 1 provides an overview of our data extraction procedure. We train our topic models using a text corpus that includes reviewer-produced inline and general comments. Our industrial partners at Dell EMC provided us with access to their code review archives from which we can extract the code review comments. To extract OpenStack Nova data, we use the Gerrit API.[2]

To mitigate the impact of noise on our topic models, we first filter out the comments that were produced by bots (e.g., integration testing bots) and replies that were written by the authors of the changes. After applying these filters, our data set contains only the comments that were written by reviewers.

Next, we identify whether the comments are natural language or code by using `NLoN`, an R package that uses machine learning to classify documents as natural language or not [27]. We then apply standard text preprocessing techniques [21] to each document, removing URLs and non-alphabetical characters, converting words to lower case, removing stop words,[3] and applying lemmatization to each token of the comment corpus. Lemmatization maps different conjugated forms of a word to their base form according to its part-of-speech tag. We use lemmatization instead of stemming [37], as it tends to better preserve term nuances [21].

2.4 Topic Modelling

Topic models are a type of statistical model that discover latent topics in a corpus of text documents. In our setting, our corpora are comprised of general and inline review comments, where each comment is represented by one document.

An Overview of LDA. We use Latent Dirichlet Allocation (LDA) [8] to detect the latent topics in the preprocessed comment corpus. Researchers have developed several topic modelling techniques for different goals [25, 43]. LDA meets our needs, as it groups discussion topics in documents [8]. LDA is probabilistic in nature and provides multiple ways to assess a topic and its related words. LDA represents topics as probability distributions over the corpus, and each word in the corpus follows a probability distribution over a topic. LDA groups words into topics using their document co-occurrence frequency. As similar-meaning words tend to co-occur more frequently than different-meaning words, words within topics are often semantically related. Thus, LDA can associate frequently co-occurring words with higher level concepts (i.e., topics).

LDA Implementation. LDA infers a topic membership distribution from the documents within an input corpus. We use the LDA implementation provided by MALLET [30], which derives LDA models based on Gibbs sampling, and is widely used within the software engineering domain [4, 22, 42].

Choice of Parameters. Training an LDA model requires settings for several hyperparameters, such as the number of topics (K), the probability of topic t in document d ($\alpha = P(t|d)$), and the probability of word w in topic t ($\beta = P(w|t)$). In MALLET, α and β can be initialized at random and automatically tuned via a re-sampling process; however, a K value must be set manually. When K is too large, topics may become fragmented and lose their semantic meaning. When K is too small, topics may become tangled and take on more than one semantic meaning. Selecting a good K value is important but is still an open research problem [11].

It has been argued that topic models should be tuned independently for different corpora [1]. Hence, we tune the LDA parameters, striving to produce a topic model that has high stability, i.e., future researchers can easily reproduce a similar topic model using our data set. To achieve model stability, we first train models with $K = [10..50]$ five times with randomly initialized α and β values. Then, for each set of models with the same K value, we calculate the R_n—a measure of the cross-run similarity of topics [1]. More specifically, R_n is the median number of occurrences of n terms that appear in all topics in all runs.

We observe that the R_n curves are the highest when $K = 10$ for both studied projects, i.e., the topics from the set of topic models that were produced with $K = 10$ share the most similarity with each other and are hence the most stable. Therefore, we use the $K = 10$ setting for our analyses.

Output of LDA. Once trained on our preprocessed data, LDA produces a set of topics that contain statistical distributions of words in the corpus. The words with higher probabilities often correspond to a related concept. For example, if the words with the highest probabilities in a topic are "log", "message", and "error", we suspect that the topic is related to logging and exception handling, and would label the topic as such.

LDA also generates a distribution of topic membership scores for any given document. More specifically, for a given document d_i, the LDA model can produce a membership score $0 \leq \delta(d_i, t_k) \leq 1$, which indicates the strength of the relationship between d_i and topic t_k (larger values indicate stronger relationships). For example, the review comment "provide more straightforward error messages and log them appropriately" will have a strong topic membership score for the logging and exception handling topic described above.

3 TOPIC PREVALENCE

Prior work has analyzed the contents of review comments. For example, Bacchelli and Bird [2] found that code reviews at Microsoft contain code improvement suggestions and requests for additional detail, in addition to addressing code defects. Mäntylä et al. [28] and Beller et al. [6] find that there are roughly three maintainability comments for every functionality one in the code reviews of several proprietary and open source systems. Prior to addressing our RQs, we set out to explore the prevalence of topics in our corpus.

3.1 Approach

We identify the high-level concepts that the LDA topics highlight by reading the 20 terms and 20 unprocessed review comments with the strongest association to each topic. We select the terms with the top 20 term weights for each topic. When ordering the terms that comprise a topic, we take inspiration from the Term Frequency Inverse Document Frequency (TF-IDF) concept. The TF is mapped to the term weight within the topic. The IDF is mapped to the Inverse Topic Frequency (ITF). We order terms by their TF-ITF score—terms with high term weight scores that appear in few other topics are considered first in our topic labelling process.

[2] https://review.openstack.org/Documentation/rest-api.html
[3] https://www.ranks.nl/stopwords

Table 2: The LDA topic themes, labels, and share values.

Theme	OpenStack Nova	Share	Sample Quote taken from OpenStack Nova	Dell EMC Project	Share
Context Specific	Volumes and Storage Management	6.1	"bdm.volume_id at this point is the old volume because we haven't updated the BDM yet, that happens on L5051. So this isn't taking into account if we failed or not. [...]"	File Locations	4.5
	Provisioning Decision Making	4.4	"[...] there is no single resource provider which has all of those resources. Instead of creating a resource provider like this, what you want to do is compute node as done in the super class and then extend its inventory to add the custom resource class. [...]"	Project Configuration	6.5
	Virtual Machine	5.1	"[...] the container setup may deny the ability to mount filesystems inside as a security restriction. [...]	Project Terminology	8.8
	API Issues	10.5	"[...] why are we making the proxy BM REST API in nova support keystone v3 when it's deprecated at the 2.36 microversion and people really shouldn't be using this proxy API anyway [...]"		
Exception Handling	Exception Handling, Logging and User-Facing Error Msg	7.3	"[...] I'd think logging a warning and not stacktracing would be sufficient since this is just a best attempt to cleanup the failed build[...]"	Logging and User-Facing Error Msg	9.3
				Exception Handling and Memory Management	8.8
Language Specific	Python Collections	8.1	"You can use the field name dict access and use collections.defaultdict(list) to reduce the complexity of the above to just this: [...]"	String/Buffer Issues	5.4
Design	Object Oriented Design	12.4	"[...] If we're going to refactor spawn then can we consider each method one-by-one and do the right thing for that method rather than pushing them all to a 'helper' class?"	Object Oriented Design	10.4
				and Concurrency Function Design	11.9
Code Review Process	Code Review Process and Minor Issues	20	"[...] I think you may have gotten the bug number wrong in the commit message?"	Code Review Process and Minor Issues	14.8
Code Style	Code Style	6.8	"[...] Per HACKING, please separate std libs from 3rd party imports with a single newline [...]"	Code Style	8.2
Unit Testing	Unit Testing	6.2	"Should there be a unittest for this function? I didn't see one. [...]"		

We also analyze the 20 review comments with the strongest association to the topic under analysis. We only include comments with the highest $\delta(d_i, t_k)$ scores for each topic t_k to avoid including comments for which the topic membership is less clear. The first and last authors collaboratively assigned labels to each topic following the procedure outlined above, both authors agreed on the final selection of all labels. As this was a discovery task, we did not independently label topics to compute an agreement score.

We apply Barua et al.'s [4] topic share metric to each topic t_k:

$$topic\_share(t_k) = \frac{1}{|D|} \sum_{\substack{\forall d_i, d_i \in D, \\ \delta(d_i,t_k) \geq 0.1/0.05}} \delta(d_i, t_k) \qquad (1)$$

where D is our corpus of comments and d_i is an individual comment. The topic share measures the proportion of documents that contains a specific topic. For example, $topic\_share(t_1) = 0.28$ indicates that 28% of the documents share a non-negligible association with t_1.

Because we train topic models with $K = 10$ topics, each topic will have a minimum membership score of $\frac{1}{10} = 0.1$ for a (theoretical) document that is not associated with any topic. On the other hand, if a document is associated with some topic t_i, the topic membership score for some other non-related topics must be less than 0.1 or 0.05. Therefore, to keep the topics that have the minimum membership score from skewing our topic share scores, we filter out topic membership scores below 0.1.

3.2 Topic Identification

Below, we provide a sample comment from two topics that share the same theme in both the Dell EMC project and in our replication on the OpenStack Nova project. In addition to these comments (and 19 other similar comments for these topics), we analyze the top 20 terms of the topics in the studied systems (40 terms total).

> Are you intentionally indenting this much space for a command that doesn't fit? Also, because all these are at the same level it makes it a bit harder to tell when commands begin and end.
> *Project: Dell EMC, topic_score(code style) = 0.96*

> No, this is correct indentation for a continued line. Visual indentation would only be 4 spaces, which would line up with the 'return' line below. Correct indentation is 2 levels, or 8 spaces.
> *Project: Nova, topic_score(code style) = 0.94*

We label these topics as *code style* because the sample comments present cosmetic issues, and the top terms are "space" and "blank".

We present the labels that authors agree upon in Table 2. The full mapping from topics to their most relevant comments is included in our replication package.[1] We include sample quotes of relevant comments for the labels of OpenStack Nova in Table 2. We cannot do the same for the Dell EMC project due to its proprietary nature.

3.3 General Observations

The proportion of review comments that are associated with context-specific topics is on par with code style or code review process discussions. Our results, presented in Table 2 show that roughly half of the comments belong to technical topics that are related to general software engineering concepts.

These results complement observations of prior work. For example, Bacchelli and Bird [2] found that although general communication is an important aspect of code reviewing practice at Microsoft, code improvement topics are more frequently discussed. Moreover, our topic models yield a similar rate of *code style* issues (6.8%–9.2%) as was observed through manual review inspection by Mäntylä and Lassenius [28] in a different context (*visual representation* concerns are raised in 9.8%–10.8% of their studied reviews).

4 LONGITUDINAL STUDY AT DELL EMC

In this section, we present the results of our study with respect to our two research questions. For each question, we first present our approach, including the measures we use to operationalize key concepts, and then present our observations.

RQ1: How does the prevalence of code review topics change as a community ages?

RQ1: Approach: To study how topic popularity changes over time, we analyze trends using the *topic impact* measure [4], which defines the impact of a topic t_k in month m as:

$$topic\_impact(t_k, m) = \frac{1}{|D(m)|} \sum_{\substack{\forall d_i, d_i \in D(m), \\ \delta(d_i, t_k) \geq 0.1}} \delta(d_i, t_k) \qquad (2)$$

where $D(m)$ is the set of review comments written in month m. In other words, topic impact measures the proportion of review comments that are associated with a topic t_k in a month m. Similar to the topic share measure, we keep negligible membership scores from skewing topic impact values by filtering out topics that have membership scores below $\frac{1}{K} = 0.1$.[4]

We first compute the topic impact score in all of the studied months for each topic. Then, we show key plots of the trends over time. We then analyze the trends of topics in both projects that belong to a similar concept. In addition to the raw values (scatterplot points), we plot a trend line using Loess-smoothed regression lines. The translucent grey shaded area shows the 95% confidence interval. We also plot vertical dashed lines at the start of each year to aid in the visual inspection of periodicity of topic impact over time.

To determine if the impact of a topic is significantly increasing or decreasing over time, we use the Cox-Stuart trend test [12] (two-tailed, $\alpha = 0.05$), which compares earlier data points to later ones.

RQ1: Results: We observe three trends in the prevalence of code review topics as communities age.

Observation 1 — Reviewers tend to provide fewer comments related to *code review process* as development teams stabilize. *Code review process* topics mainly include comments that address issues related to procedural formalities and minor issues of the code review process (e.g., correctly formatting the commit message, or referencing the correct bug number). We observe that the trend of this topic in the Dell EMC project is non-monotonic, slightly increasing before late 2015 and decreasing afterwards, as shown in Figure 2. We discussed this with a development manager from the Dell EMC project who explained that the period with the steadiest growth (early/mid 2015) coincides with a large change in the composition of the development team. During this period, there was a large influx of new members into the development team. This may explain why more comments appear with respect to code review process in that period. As the new members became more familiar with the code review process, the topic trend begins to descend (early 2016).

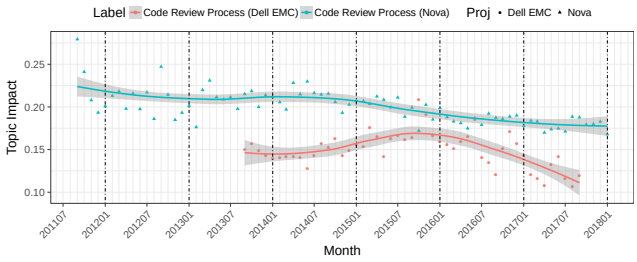

Figure 2: The impact score of topics discussing *Code Review Process* issues as the studied projects age (RQ1).

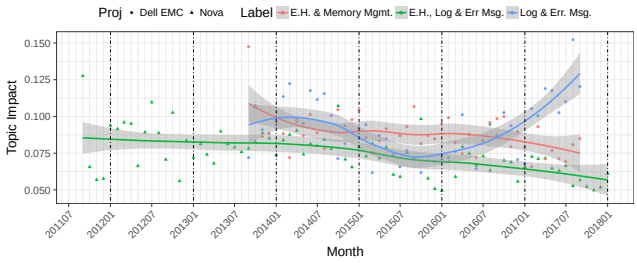

Figure 3: *Exception Handling* topics over time (RQ1).

Observation 2 — While *exception handling and memory management* shows a downward trend, there is a shift towards an upward trend for the *logging and user-facing error message* topic in late 2015. There are two Dell EMC topics on *Exception Handling*. One is related to exception handling and memory management, and the other is related to logging and user-facing error messages (Figure 3). The Dell EMC project is primarily written in C. Hence, members in the Dell EMC community discuss exception handling issues related to memory management frequently enough for our LDA models to create a distinct topic in this theme.

Dell EMC developers and operators use system logs to debug software components and diagnose and recover from issues at runtime. Thus, it is crucial that log messages are clear and concise. The trough and subsequent growth in logging and memory management coincides with the same influx of new developers (cf. Observation 1), suggesting that the change of group dynamics could also affect the choice of discussion topics in the community.

[4] A sensitivity analysis that explores threshold values up to 0.2 shows no significant change in topic trends. See the replication package for more detail.

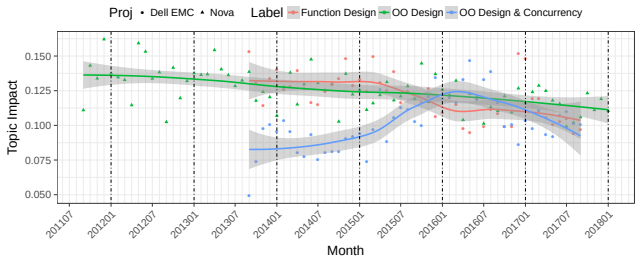

Figure 4: *Design* topics over time (RQ1).

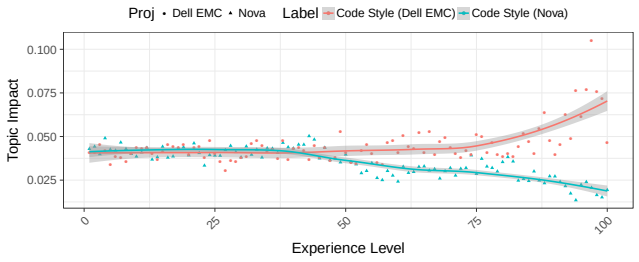

Figure 5: The impact score of topics discussing *Code Style* issues as the reviewers accrue experience (RQ2).

Observation 3 — Although the *function design* topic has a decreasing trend, the *object-oriented design and concurrency* topic has a trend that shifted directions in Q1 2016. Two Dell EMC topics share the *design* theme. Since design is crucial to the structure of a system, seeing that design-related topics are showing a decreasing trend in Figure 4 may signal to community members that an appraisal is needed. For example, the *function design* topic has a consistent downward trend.

Figure 4 also shows that in early 2016, the *object-oriented design and concurrency* topic shifts from an increasing trend to decreasing one. In addition, the decreasing *function design* trend flattens out. A manager at Dell EMC explained that these trend changes coincide with a shift of a number of highly active staff members from an area of the codebase that is primarily implemented in object-oriented languages like C++ and Java to another area that is primarily implemented in procedural languages like C.

RQ1: *The topics covered by reviewing feedback are continually evolving at Dell EMC. Topic models can highlight these evolving trends, which often coincide with project events of significance.*

RQ2: How does the prevalence of code review topics change as reviewers accrue experience?

RQ2: Approach: We analyze the relationship between reviewer experience and the prevalence of topics in their review feedback by (1) using heuristics to estimate reviewer experience and (2) calculating the topic impact over different experience levels.

Reviewer Experience Heuristic by Number of Reviews. We use the number of prior comments to estimate reviewer experience. For example, a reviewer who has written 100 comments prior to writing comment C has an experience of 100 when C was written. This assumes that reviewers gain experience as they write comments. Mockus and Herbsleb [33] found a link between developer expertise and the number of changes that a developer has written. Thongtanunam et al. [44] showed that the concept extends to reviewing activity. Thus, we believe that our assumption is sound.

To avoid oversampling the experience signal, we group experience scores into 100 levels. According to our heuristic, a reviewer's experience score (slightly) grows after each comment that they write. Since the experience score is heavily right skewed (i.e., there are far more inexperienced reviewers than experienced ones), grouping experience values using equidistant thresholds will undersample the low experience values and oversample the high experience values. Therefore, we group them into 100 bins that contain an equal number of reviewers. We select 100 as the number of bins by analyzing the distribution of the data. Analysis of the data set with 50 and 75 bins suggests that the general direction and shape of the distribution remain consistent. We include the figures of the 50- and 75-bin experiments in our replication package.[1]

Topic Impact by Experience Levels. Similar to the topic popularity analysis of RQ1, we compute *topic_exp_impact*—a measure of topic prevalence for reviewers with a given level of experience. We redefine the topic impact measure (Equation 2) to measure the experience impact of a topic t_k in an experience level x:

$$topic\_exp\_impact(t_k, x) = \frac{1}{|D(x)|} \sum_{\substack{\forall d_i, d_i \in D(x), \\ \delta(d_i, t_k) \geq 0.1}} \delta(d_i, t_k) \quad (3)$$

where $D(x)$ is the set of comments written by reviewers with experience level x and *topic_exp_impact* is the proportion of review comments that have a non-negligible association with topic t_k (i.e., $\delta(d_i, t_k) \geq 0.1$) for reviewers with experience level x.[5]

For our experience level analysis, we show key plots of the trends. Similar to RQ1, the plots contain the data points, Loess-smoothed regression lines, and shaded 95% confidence intervals. We also apply Cox-Stuart tests to each of the trends.

RQ2: Results: We make three observations about the prevalence of code review topics as reviewers accrue experience.

Observation 4 — As reviewers accrue experience, the trend in the *code style* topic increases. Figure 5 shows that after a stable period, the experience trend for *code style* increases in the Dell EMC project. On the surface, code style feedback seems to provide a low return on investment, and having experienced reviewers spend their effort on code style seems wasteful. Thus, the increasing trend may raise concerns for Dell EMC management. However, in a follow-up meeting, a Dell EMC manager explained that senior staff had raised concerns about code style problems in recent team meetings. Hence, the rate at which code style concerns are being raised increases with experience agrees with his first-hand experience.

Observation 5 — *OO design and concurrency* and *function design* feedback tends to decrease as reviewers gain experience. Figure 6 shows that as reviewers gain more experience, they

[5] Similar to RQ1, a sensitivity analysis of thresholds of 0.1–0.2 yielded no significant change in topic trends. See the replication package for more detail.

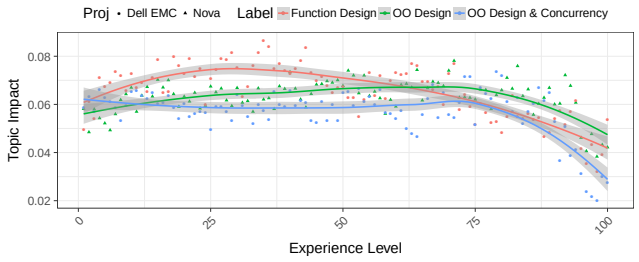

Figure 6: The impact score of topics discussing *Design* issues as the reviewers accrue experience (RQ2).

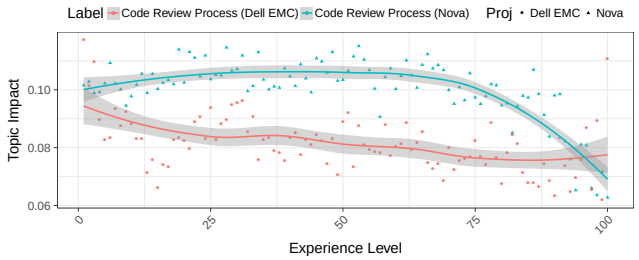

Figure 7: The impact score of topics discussing *Code Review Process* issues, plotted with regard to reviewing experience.

tend to discuss design issues less often. Since *function design* covers low level issues, reviewers with less project experience are able to provide this context-agnostic type of feedback. Meanwhile, *OO design and concurrency* only sees a sharp drop in discussion at the upper experience levels. This may signal that while this topic may not be as accessible as lower-level *function design* feedback, it still does not require an expert.

Observation 6 — The *code review process* topic tends to decrease as reviewers accrue experience. Figure 7 shows a downward trend of topic impact in *code review process* discussions for Dell EMC reviewers when they become more experienced. Developers at Microsoft [9] and Mozilla [23] argue that when trying to improve their changes, code review process comments are not as helpful as more context-specific comments. The observed trend for the *code review process* topic of Dell EMC suggests that most of the experienced reviewers indeed follow this trend and provide fewer process-related details in their review feedback.

RQ2: *Reviewing behaviour also evolves are reviewers accrue experience. Topic models show that experienced reviewers specialize in different ways according to the needs of their community.*

5 OPENSTACK NOVA REPLICATION STUDY

In this section, we describe the results of our replication on OpenStack Nova as a comparison to the Dell EMC results.

RQ1: How does the prevalence of code review topics change as a community ages?

Observation 1 — OpenStack Nova reviewers also tend to provide fewer comments related to *code review process* over time. While Dell EMC presents a non-monotonic trend for this topic, Figure 2 shows a consistent decreasing trend of this topic in Nova. The explicit Nova development guidelines,[6] which have been refined over time, may be contributing to this decrease.

Observation 2 — Fewer *exception handling* topics emerge in Nova than Dell EMC, but trends in both communities are decreasing. Since Nova is primarily written in Python, a language with garbage collection built in, memory management is generally not an issue that developers need to discuss. Hence, while two *exception handling* topics emerged for Dell EMC we only observe one topic in Nova (i.e., E.H., Log & Err Msg.). Figure 3 shows that the Nova *exception handling* topic is also decreasing over time.

Similarly to Dell EMC, it is crucial for OpenStack Nova that system log messages are clear and concise. The decreasing trends in both organizations may indicate that either the community has become more adept at logging and exception handling over time, or that reviewers have put less effort into raising these concerns in more recent time periods. In either case, the topic models can be used to identify and monitor such trends.

Observation 3 — Similar to the *function design* topic in the Dell EMC project, the *object-oriented design* topic in Nova is showing a consistent downward trend. While two *design* topics emerge at Dell EMC, only one such topic exists for Nova: *object-oriented design*. Similar to Dell EMC, the downward trend in *design* presented in Figure 4 may signal to community members that an appraisal is needed.

RQ1: *Although OpenStack and Dell EMC present different communities with independent timelines, topic models can be used to detect similar trends as their communities age.*

RQ2: How does the prevalence of code review topics change as reviewers accrue experience?

Observation 4 — Contrary to Dell EMC reviewers, as Nova reviewers accrue experience, the trend in the *code style* topic decreases. As previously discussed, having experienced reviewers spend their effort on code style may be wasteful. Thus, although the Dell EMC trend may have been informed by senior staff concerns, the decreasing trend shown in Figure 5 for Nova is encouraging.

Observation 5 — Similar to *function design* in Dell EMC, we observe a decreasing trend for *OO design and concurrency* in Nova. Figure 6 shows that as reviewers gain more experience, the *object-oriented design* topic in Nova is less frequently raised. Since the object-oriented programming skills are required for most Nova developers, the barrier to entry for this type of comment is lower than other topics. Thus, reviewers with less project experience may find it easier to raise context-agnostic *object-oriented design* issues.

Observation 6 — The *code review process* topic also tends to decrease for Nova as reviewers accrue experience. The observed trends in Figure 7 for the *code review process* topics of both

[6] https://github.com/openstack/nova/blob/master/HACKING.rst

studied projects suggest that most of the experienced reviewers indeed provide fewer process-related details in their review feedback.

RQ2: *Although different communities may have different concerns, reviewing behaviour does evolve as reviewers accrue experience.*

6 THREATS TO VALIDITY

External validity. Threats to external validity have to do with the generalizability of our results. We conduct an empirical study of the Dell EMC community. Although we replicate our approach on OpenStack, our results may not generalize to other settings. Since OpenStack is open source, the community is comprised of a broad spectrum of developers, including hobbyists and professionals. Moreover, the professional developers represent several companies, each with its own corporate culture (e.g., IBM, NEC). Although we only analyze two communities, the sample of developers and development cultures is rich and diverse, covering a broad range of developer backgrounds.

Internal validity. Threats to internal validity have to do with whether other plausible hypotheses could explain our results. Our work focuses on comments that are recorded in code review platforms as an indicator of the discussions that take place during code review. Although the Dell EMC and OpenStack communities tightly integrate their code review platforms into their development cycle, in-person meetings or other communication media (e.g., e-mail) could be used to supplement the discussions that are happening on the platform. Unfortunately, explicit links between code changes and other communication channels are scarce, and recovering these links is a non-trivial research problem [3, 7].

We assume that changes in topic impact are due to changes in time period or reviewer experience; however, confounding factors may play a role. For example, the size of the backlog of tasks that a developer is working on may also impact the type of feedback that they provide. An initial exploration of the reviewer backlog indicates that it shares a high correlation with reviewer experience. Preliminary analyses that we performed to study the influence of reviewer workload yields near-identical insights as our experience analysis in RQ2. Nonetheless, we plan to explore this and other potential confounding factors, such as the phase of project development and the features being reviewed in future work.

Construct validity. Threats to construct validity have to do with the alignment of our choice of indicators with what we set out to measure. Our experience heuristics are based on the number of prior review comments in either dataset. Reviewers may gain experience from other channels, e.g., development and review activities in other projects. Nevertheless, we use an experience heuristic that we can measure using the data that we have on hand.

To conduct our experiment, we need to select settings for: (1) the number of topics in our LDA model ($K = 10$), (2) the time units for RQ1 (months), and (3) the number of experience levels for RQ2 (100). We have experimented with tuning parameters and maximized the stability of our topic model; however, selecting different settings may still yield different results. The goal of this study is not to identify the optimal settings for these parameters, but rather to examine whether and how reviewing feedback changes as communities age and reviewers accrue experience.

7 RELATED WORK

In this section, we discuss the related work with respect to code review and topic modelling in software engineering.

Code Review. The proliferation of code review data, and tools for analyzing it, have made several recent studies possible. Several papers have shared data sets of (and tools for interfacing with) Gerrit repositories [18, 34]. Since these data sets can be large and difficult to understand, tools like ReDA [46] and Bicho [17] support the analysis of review data. In the same spirit of openness, we share the data from our analysis of OpenStack Nova.[1]

Code review is more than an exercise in defect hunting. Code review also serves as a platform for knowledge transfer and collaborative problem solving [2]. Rigby and Storey [41] analyzed interactions in the code review processes of five open source systems and found that in addition to defect prevention, developers also talk about features, scope, and process issues. Baysal et al. [5] performed an empirical study on WebKit and found the developer's affiliation and level of participation influence the outcome of code review. Mäntylä and Lassenius [28] and Beller et al. [6] found that review discussions raise and fix three maintainability issues for every functional issue. Similar to prior work, we also analyze the rich data stored in code reviewing archives; however, we set out to better understand how reviewing feedback changes as communities age and reviewers accrue experience.

Reviewer experience is a crucial factor that affects the value derived from review comments. Bacchelli and Bird [2] analyzed code review comments at Microsoft, and report that more prior knowledge of the code triggers more valuable feedback. Bosu et al. [9] analyzed the usefulness of review comments at Microsoft, and found that reviewers with greater seniority tend to provide the more useful comments. Rigby et al. [40] found that open source contributors with more expertise in code review are the ones who provide context-specific feedback. Di Biase et al. [13] found that reviews that were conducted by multiple reviewers tend to be more successful at finding security issues. Rahman et al. [38] developed *RevHelper*, which helps developers discover whether their code review comments are useful through a prediction model. To aid in improving reviewing skill, we demonstrate the potential of and lay the foundation for tool support for exploring personal and team reviewing trends and tendencies.

Recent studies focus on different types of code review comments. Pangsakulyanont et al. [36] used semantic similarity to group 72,000 review comments into different topics, and found that most code review comments are often unrelated to defect prevention, with some of them discussing trivial issues. Kononenko et al. [24] observed that 54% of reviewed changes are bug-inducing, and concluded that code reviewers tend to miss bugs. Kononenko et al. [23] surveyed 88 core Mozilla developers and found that review quality is mainly associated with the thoroughness of the feedback. Moreover, reviewers find it difficult to maintain their technical skillset for writing high-quality reviews. Norikane et al. [35] conjectured that different kinds of code review feedback affect the willingness of a volunteer contributor to engage in open source software. Zhu et al. [52] showed that improvements in code review management, e.g., providing clearer guidelines for reviewers, make contribution processes more efficient. The topic models in this paper can support

a community and reviewer analytics dashboard that would enable data-grounded management of code review practices.

Topic Modelling. Topic models have been used in various software engineering experiments. Xia et al. [49] used topic models to recommend tags to describe the most important features of posted content or projects. Zhao et al. [51] used LDA to extract topics from discussions involving bug fixes in five open source projects, and explore the relationship between the frequency of discussion and bug reworking. Maskeri et al. [29] used LDA to extract business topics from identifiers and comments in source code. We use LDA to extract and identify topics from code review comments.

In addition to identifying topics from a text corpus, previous work has also analyzed trends in emergent topics. Barua et al. [4] used LDA to study how technical topics and programming languages on StackOverflow change over time. Linstead et al. [26] applied LDA to source code and investigate the evolution of programming concepts as codebases grow. Hindle et al. [20] applied topic models on developer communication corpora within predefined time windows, and visualize the topics and their trends over time according to those windows. We focus on changes in emergent topics over time and as developers accrue experience.

Since topic models have achieved broad adoption in software engineering research, prior work also points out pitfalls and suggests solutions. Chen et al. [11] surveyed 167 software engineering papers that use topic models, pointing out pitfalls, such as interpretation issues and a lack of exploration of the parameter space. They suggest software engineering researchers to keep up with the machine learning community while applying topic modelling techniques. Hindle et al. [19] surveyed developers and project managers about how they interpret topics generated by topic models, finding that the level of difficulty for interpretation varies across topics. We recognize the importance of topic model construction and validation. We rely on work from the natural language processing community [10, 50] to support us when training our models. When interpreting our models, we solicit feedback from staff at Dell EMC, who were able to provide valuable insights and point to coinciding events to explain shifting trends.

8 CONCLUSIONS

To derive value from a code review process, it must produce valuable feedback. While past work [6, 28] has explored the issues found and fixed during code review, little is known about how review feedback changes as communities age and its members accrue experience.

In this paper, we train topic models to identify latent topics in the review comments at Dell EMC. To evaluate the generalizability of our approach and provide an open replication package,[1] we replicate our analysis on OpenStack Nova. Through a study of 248,695 review comments in 39,249 changes, we observe that (1) context-specific issues (e.g., API-related topics) are more frequently discussed than formatting issues; (2) changes in the reviewing behaviour of a code review community often coincides with project events of significance (e.g., large changes in team composition); and (3) common trends in the topics that reviewers raise as they accrue experience are rare, since they often tailor their feedback to the needs of the communities that they serve. In our estimation, the observations from our study have two key implications.

Monitoring Community and Reviewer Skill Development. We believe that our topic models lay the necessary groundwork for a reviewing feedback analytics framework. We observe that the prevalent topics in code reviewing discussions change as a review community evolves (RQ1). A tool that analyzes quantitative data from topic models like ours could be used to track changes in community and reviewer focus in a cost-effective, explainable, and easily replicable manner. These changes can be compared with qualitative data that is gathered from the development team to detect whether the reviewing process is keeping up with community expectations. In addition, reviewers could use such a low upkeep system to monitor whether their personal feedback trends are meeting their own reviewing goals.

Mentorship Programs in Code Review. Our findings suggest that there is a gap between novice and experienced reviewers. Results from RQ2 show that the prevalent topics in code reviewing discussions differ across experience levels. Knowledge transfer is one of the main benefits of code review [2]. An explicit mentorship program may help novice reviewers to hone their reviewing skills more quickly. By receiving additional knowledge about the feedback that experienced reviewers provide, the novice could accumulate experience and produce more useful review comments more quickly, without the need to accrue months or years of less effective reviewing experience. For example, by providing novice reviewers with trends and code review examples produced by experienced reviewers, we can raise awareness of what a high quality review looks like. This could take the form of constructive feedback like "more experienced reviewers tend to discuss Code Review Process issues less often, so try to put less emphasis on this in your future reviews".

The key contribution of this study is the evidence of significant trends in reviewing behaviour and a set of measures that can be tracked as communities age and their stakeholders accrue experience. More specifically, based on trends in emergent topics, researchers and developers can build tools to suggest directions or identify blind spots in the reviewing behaviour of communities and individuals. For example, in future work, we plan to use our topic models to build an analytics dashboard for monitoring reviewing behaviour. Extracting reviewing behavior using our topic models may help to better focus on topics of higher importance, yielding a review process that generates more value for its community.

REFERENCES

[1] Amritanshu Agrawal, Wei Fu, and Tim Menzies. 2018. What is wrong with topic modeling? And how to fix it using search-based software engineering. *Information and Software Technology* (2018).

[2] Alberto Bacchelli and Christian Bird. 2013. Expectations, outcomes, and challenges of modern code review. In *Proceedings of the International Conference on Software Engineering (ICSE)*. IEEE, 712–721.

[3] Alberto Bacchelli, Michele Lanza, and Romain Robbes. 2010. Linking e-mails and source code artifacts. In *Proceedings of the International Conference on Software Engineering (ICSE)*. ACM, 375–384.

[4] Anton Barua, Stephen W Thomas, and Ahmed E Hassan. 2014. What are developers talking about? an analysis of topics and trends in stack overflow. *Empirical Software Engineering* 19, 3 (2014), 619–654.

[5] Olga Baysal, Oleksii Kononenko, Reid Holmes, and Michael W Godfrey. 2013. The influence of non-technical factors on code review. In *Proceedings of the Working Conference on Reverse Engineering (WCRE)*. IEEE, 122–131.

[6] Moritz Beller, Alberto Bacchelli, Andy Zaidman, and Elmar Juergens. 2014. Modern code reviews in open-source projects: Which problems do they fix?. In *Proceedings of the Working Conference on Mining Software Repositories (MSR)*. ACM, 202–211.

[7] Christian Bird, Alex Gourley, and Prem Devanbu. 2007. Detecting patch submission and acceptance in oss projects. In *Proceedings of the International Workshop on Mining Software Repositories (MSR)*. IEEE, 26–26.

[8] David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. *Journal of Machine Learning Research* 3, Jan (2003), 993–1022.

[9] Amiangshu Bosu, Michaela Greiler, and Christian Bird. 2015. Characteristics of useful code reviews: An empirical study at microsoft. In *Proceedings of the Working Conference on Mining Software Repositories (MSR)*. IEEE, 146–156.

[10] Jonathan Chang, Jordan L Boyd-Graber, Sean Gerrish, Chong Wang, and David M Blei. 2009. Reading tea leaves: How humans interpret topic models.. In *Nips*, Vol. 31. 1–9.

[11] Tse-Hsun Chen, Stephen W Thomas, and Ahmed E Hassan. 2016. A survey on the use of topic models when mining software repositories. *Empirical Software Engineering* 21, 5 (2016), 1843–1919.

[12] David Roxbee Cox and Alan Stuart. 1955. Some quick sign tests for trend in location and dispersion. *Biometrika* 42, 1/2 (1955), 80–95.

[13] Marco di Biase, Magiel Bruntink, and Alberto Bacchelli. 2016. A security perspective on code review: The case of Chromium. In *Proceedings of the International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 21–30.

[14] Michael E Fagan. 2001. Design and code inspections to reduce errors in program development. In *Pioneers and Their Contributions to Software Engineering*. Springer, 301–334.

[15] Daniel M German, Gregorio Robles, Germán Poo-Caamaño, Xin Yang, Hajimu Iida, and Katsuro Inoue. 2018. Was my contribution fairly reviewed? A framework to study fairness in Modern Code Reviews. In *Proceedings of the International Conference on Software Engineering (ICSE)*. ACM, 523–534.

[16] Jesus M. Gonzalez-Barahona, Daniel Izquierdo-Cortazar, Stefano Maffulli, and Gregorio Robles. 2013. Understanding How Companies Interact with Free Software Communities. *IEEE Software* 30, 5 (2013), 38–45.

[17] Jesus M. Gonzalez-Barahona, Daniel Izquierdo-Cortazar, Gregorio Robles, and Alvaro del Castillo. 2014. Analyzing Gerrit Code Review Parameters with Bicho. In *Proceedings of the International Workshop on Software Quality and Maintainability (SQM)*. 1–12.

[18] Kazuki Hamasaki, Raula Gaikovina Kula, Norihiro Yoshida, Ana Erika Camargo Cruz, Kenji Fujiwara, and Hajima Iida. 2013. Who Does What during a Code Review? Datasets of OSS Peer Review Repositories. In *Proceedings of the Working Conference on Mining Software Repositories (MSR)*. 49–52.

[19] Abram Hindle, Christian Bird, Thomas Zimmermann, and Nachiappan Nagappan. 2015. Do topics make sense to managers and developers? *Empirical Software Engineering* 20, 2 (2015), 479–515.

[20] Abram Hindle, Michael W Godfrey, and Richard C Holt. 2009. What's hot and what's not: Windowed developer topic analysis. In *Proc. of the Int'l Conference on Software Maintenance (ICSM)*. IEEE, 339–348.

[21] Dan Jurafsky. 2000. *Speech & language processing*. Pearson Education.

[22] Nafiseh Kahani, Mojtaba Bagherzadeh, Juergen Dingel, and James R Cordy. 2016. The problems with eclipse modeling tools: a topic analysis of eclipse forums. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*. ACM, 227–237.

[23] Oleksii Kononenko, Olga Baysal, and Michael W Godfrey. 2016. Code review quality: how developers see it. In *Proceedings of the International Conference on Software Engineering (ICSE)*. ACM, 1028–1038.

[24] Oleksii Kononenko, Olga Baysal, Latifa Guerrouj, Yaxin Cao, and Michael W Godfrey. 2015. Investigating code review quality: Do people and participation matter?. In *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 111–120.

[25] Victor Lavrenko and W Bruce Croft. 2001. Relevance based language models. In *Proceedings of the International Conference on Research and Development in Information Retrieval*. ACM, 120–127.

[26] Erik Linstead, Cristina Lopes, and Pierre Baldi. 2008. An application of latent Dirichlet allocation to analyzing software evolution. In *Proceedings of the International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 813–818.

[27] Mika V Mäntylä, Fabio Calefato, and Maelick Claes. 2018. Natural Language or Not (NLoN) - A Package for Software Engineering Text Analysis Pipeline. In *Proc. of the 15th International Conf. on Mining Software Repositories (MSR)*. to appear.

[28] Mika V Mäntylä and Casper Lassenius. 2009. What types of defects are really discovered in code reviews? *Transactions on Software Engineering (TSE)* 35, 3 (2009), 430–448.

[29] Girish Maskeri, Santonu Sarkar, and Kenneth Heafield. 2008. Mining business topics in source code using latent dirichlet allocation. In *Proceedings of the India Software Engineering Conference*. ACM, 113–120.

[30] Andrew Kachites McCallum. 2002. MALLET: A Machine Learning for Language Toolkit. (2002). http://mallet.cs.umass.edu.

[31] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E Hassan. 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 Working Conference on Mining Software Repositories (MSR)*. ACM, 192–201.

[32] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E Hassan. 2016. An empirical study of the impact of modern code review practices on software quality. *Empirical Software Engineering* 21, 5 (2016), 2146–2189.

[33] Audris Mockus and James D Herbsleb. 2002. Expertise browser: a quantitative approach to identifying expertise. In *Proceedings of the International Conference on Software Engineering (ICSE)*. ACM, 503–512.

[34] Murtuza Mukadam, Christian Bird, and Peter C. Rigby. 2013. Gerrit Software Code Review Data from Android. In *Proc. of the 10th Working Conf. on Mining Software Repositories (MSR)*. 45–48.

[35] Takuto Norikane, Akinori Ihara, and Kenichi Matsumoto. 2017. Which review feedback did long-term contributors get on OSS projects?. In *Proceedings of the International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 571–572.

[36] Thai Pangsakulyanont, Patanamon Thongtanunam, Daniel Port, and Hajimu Iida. 2014. Assessing MCR discussion usefulness using semantic similarity. In *Proceedings of the International Workshop on Empirical Software Engineering in Practice (IWESEP)*. IEEE, 49–54.

[37] Martin F Porter. 1980. An algorithm for suffix stripping. *Program* 14, 3 (1980), 130–137.

[38] Mohammad Masudur Rahman, Chanchal K Roy, and Raula G Kula. 2017. Predicting usefulness of code review comments using textual features and developer experience. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 215–226.

[39] Peter C Rigby and Christian Bird. 2013. Convergent contemporary software peer review practices. In *Proceedings of the Joint Meeting on Foundations of Software Engineering (FSE)*. ACM, 202–212.

[40] Peter C Rigby, Daniel M German, Laura Cowen, and Margaret-Anne Storey. 2014. Peer review on open-source software projects: Parameters, statistical models, and theory. *Transactions on Software Engineering and Methodology (TOSEM)* 23, 4 (2014), 35.

[41] Peter C Rigby and Margaret-Anne Storey. 2011. Understanding broadcast based peer review on open source software projects. In *Proceedings of the International Conference on Software Engineering (ICSE)*. ACM, 541–550.

[42] Christoffer Rosen and Emad Shihab. 2016. What are mobile developers asking about? a large scale study using stack overflow. *Empirical Software Engineering* 21, 3 (2016), 1192–1223.

[43] Yee Whye Teh, Michael I Jordan, Matthew J Beal, and David M Blei. 2004. Sharing Clusters among Related Groups: Hierarchical Dirichlet Processes.. In *Nips*. 1385–1392.

[44] Patanamon Thongtanunam, Shane McIntosh, Ahmed E. Hassan, and Hajimu Iida. 2016. Revisiting Code Ownership and Its Relationship with Software Quality in the Scope of Modern Code Review. In *Proceedings of the International Conference on Software Engineering (ICSE)*. 1039–1050.

[45] Patanamon Thongtanunam, Chakkrit Tantithamthavorn, Raula Gaikovina Kula, Norihiro Yoshida, Hajimu Iida, and Ken-ichi Matsumoto. 2015. Who should review my code? A file location-based code-reviewer recommendation approach for modern code review. In *Proceedings of the International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 141–150.

[46] Patanamon Thongtanunam, Xin Yang, Norihiro Yoshida, Raula Gaikovina Kula, Ana Erika Camargo Cruz, Kenji Fujiwara, and Hajimu Iida. 2014. ReDA: A Web-based Visualization Tool for Analyzing Modern Code Review Dataset. In *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*. 605–608.

[47] Jason Tsay, Laura Dabbish, and James Herbsleb. 2014. Let's talk about it: evaluating contributions through discussion in GitHub. In *Proceedings of the International Symposium on Foundations of Software Engineering (FSE)*. ACM, 144–154.

[48] Perry van Wesel, Bin Lin, Gregorio Robles, and Alexander Serebrenik. 2017. Reviewing Career Paths of the OpenStack Developers. In *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*. 543–548.

[49] Xin Xia, David Lo, Xinyu Wang, and Bo Zhou. 2013. Tag recommendation in software information sites. In *Proceedings of the Working Conference on Mining Software Repositories (MSR)*. IEEE, 287–296.

[50] Weizhong Zhao, James J Chen, Roger Perkins, Zhichao Liu, Weigong Ge, Yijun Ding, and Wen Zou. 2015. A heuristic approach to determine an appropriate number of topics in topic modeling. *BMC Bioinformatics* 16, 13 (2015), S8.

[51] Yu Zhao, Feng Zhang, Emad Shihab, Ying Zou, and Ahmed E Hassan. 2016. How Are Discussions Associated with Bug Reworking?: An Empirical Study on Open Source Projects. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement (ESEM)*. ACM, 21:1–21:10.

[52] Jiaxin Zhu, Minghui Zhou, and Audris Mockus. 2016. Effectiveness of code contribution: from patch-based to pull-request-based tools. In *Proceedings of the International Symposium on Foundations of Software Engineering (FSE)*. ACM, 871–882.