On potential improvements in the analysis of the evolution of themes in code review comments

Umar Iftikhar, Jürgen Börstler, and Nauman bin Ali
Blekinge Institute of Technology, Karlskrona, Sweden
{umar.iftikhar, jurgen.borstler, nauman.ali}@bth.se

***Abstract*—*Context:* The modern code review process is considered an essential quality assurance step in software development. The code review comments generated can provide insights regarding source code quality and development practices. However, the large number of code review comments makes it challenging to identify interesting patterns manually. In a recent study, Wen et al. used traditional topic modeling to analyze the evolution of code review comments. Their approach could identify interesting patterns that may lead to improved development practices.**

***Objective:* In this study, we investigate potential improvements to Wen et al.'s state-of-the-art approach to analyze the evolution of code review comments.**

***Method:* We used 209,166 code review comments from three open-source systems to explore and empirically analyze alternative design and implementation choices and demonstrate their impact.**

***Results:* We identified the following potential improvements to the current state-of-the-art as described by Wen et al.: 1) utilize a topic modeling method that is optimized for short texts, 2) a refined approach for identifying a suitable number of topics, and 3) a more elaborate approach for analyzing topic evolution. Our results indicate that the proposed changes have quantitatively different results than the current approach. The qualitative interpretation of the topics generated with our changes indicates their usefulness.**

***Conclusions:* Our results indicate the potential usefulness of changes to state-of-the-art approaches to analyzing the evolution of code review comments, with practical implications for researchers and practitioners. However, further research is required to compare the effectiveness of both approaches.**

I. Introduction

Modern code reviews are a common step in software quality assurance [1–3]. Apart from improving software quality, modern code reviews also support sharing knowledge with developers new to a code base [4, 5]. The reviewer comments provided in code reviews may provide useful information for system development and evolution. Studies have shown that approximately 75% of the issues discussed in code review comments relate to improving the maintainability of the software and 25% of the feedback relates to improving its functionality [6, 7]. From the perspective of project and quality managers, it would be interesting to identify common themes within the code review comments and to analyze how these themes evolve over time. Such an analysis could help project and quality managers to understand theme evolution and aid them in introducing systematic improvements, improving company-wide development guidelines [8], or identifying training needs for developers.

The number of code commits and associated code reviews in medium to large projects can lead to hundreds of code review comments [5]; thus, manually analyzing the evolution of common themes in code review comments is infeasible. Previous studies have manually identified quality-related themes in a small set of code review comments [6, 7]. Automatic classification of code review comments is an open research question, and recent studies have also used machine learning approaches with promising results [8–11]. Recently, Wen et al. [11] demonstrated that traditional topic modeling approaches can be used to identify common themes within code review comments and to analyze their evolution over time.

In this study, we analyze the design and implementation choices in Wen et al.'s approach to studying the evolution of common themes in code review comments. We started by replicating the research design used by Wen et al. to propose potential alternatives to their approach. We then empirically analyzed each alternative to demonstrate its potential impact.

The remainder of the paper is organized as follows. In Section II, we discuss related works, whereas in Section III, we cover the adopted methodology. Section IV presents our findings, followed by a discussion of the results (Section V) and a discussion of threats to validity (Section VI). Section VII concludes the paper.

II. Related work

Several studies have manually analyzed code review comments. Mäntlya et al. [6] manually analyzed nine industrial and 23 academic systems, classified code defects discussed in code review, and proposed a taxonomy of defects discovered in code reviews. Beller et al. [7] manually analyzed two open-source systems and studied 1400 code changes in code review to identify fixed code issues, thus demonstrating the practical benefits of the code review process. Gunawardena et al. [12] provided a fine-grained taxonomy of defects discussed in code review comments by manually analyzing 417 code review comments. They further identify which code defects can be resolved using existing static analysis tools to reduce the overall effort required in the modern code review process. However, none of these studies utilize an automated method to analyze code review comments and thus are time intensive to implement.

In contrast, other studies have explored the feasibility of automating code review comments. Tufano et al. [10] utilized deep learning to select candidate code review comments from the repository of code reviews for a given code commit with up to 31% accuracy. Hong et al. [13] improved accuracy to 42% using a retrieval-based code review comments tool. This was further improved by Zampetti et al. [14], who used cosine similarity between code review comments and CheckStyle warning descriptions. They presented an automated approach for configuring the CheckStyle [1] static analysis tool, achieving a precision of 61% and a recall of 52%. However, the scope of these studies is towards automation using code review comments for automatic code review comment generation or configuring static analysis tools, which differs from our goal of analyzing code review comments.

Arafat et al. [15] used supervised machine learning algorithms to categorize and predict the topics in code review comments from six closed-source systems and reported 63% accuracy for Support Vector Machine (SVM) method. However, their approach needs a manually labeled dataset for the initial training.

Ochodek et al. [8] utilized the Bidirectional Encoder Representation from Transformers (BERT) language model to automatically classify 2,672 code review comments from three open-source systems to classify discussed topics within code review comments and achieved an average accuracy of over 80% when compared to manually classified code review comments. However, their study only considers a small set of 2,672 code review comments, and its effectiveness on a large dataset is yet to be evaluated.

Wen et al. [11] investigated how community and personal feedback trends evolve as the community matures using topic modeling. They utilized Latent Dirichlet Allocation (LDA) on one open-source system, Nova, and one closed-sourced system to study the evolution of themes in code review comments from 2011 and 2018. They considered topic stability [16] over five runs of LDA to select a suitable number of topics and assessed values between N=[10..50] as the range to search for a suitable number of topics. Their results show that the context-specific and technical feedback increases with the community's maturity and improved reviewer experience. Our work extends their study by identifying decision points where other alternatives may lead to better results. Each potential alternative proposed in this study is also demonstrated by using code review comments from three open-source systems.

Silva et al. [17] surveyed how different topic modeling methods have been used for various tasks in software engineering. Qiang et al. [18] provided a taxonomy of short-text topic modeling approaches along with an open-source Java implementation of the studied topic modeling methods and compared the performance of eight topic modeling methods on six datasets. Their results show that Dirichlet multinomial mixture (DMM) based models perform best on all considered datasets. We used their survey to identify candidate techniques to consider instead of LDA as used by Wen et al. [11].

[1]https://checkstyle.org/

III. Methodology

As stated in Section I, our study explores potential improvements of the promising approach developed by Wen et al. [11] to analyze common themes discussed by reviewers using topic modeling. To achieve the stated study aims, we define the following research question:

RQ1: ***How can we improve state-of-the-art approaches to study the evolution of code review comments?***

RQ2: ***Which common themes in code review comments can we identify using the suggested modifications to Wen et al.'s approach and how do these themes evolve over time?***

Using Wen et al.'s [11] research design as a baseline, we investigate possible improvements in the choice of algorithm, the strategy of selecting a suitable number of topics, and the approach for analyzing topic evolution. Specifically, we compare the topic stability of topics generated by traditional topic modeling used by Wen et al. [11] and the short-text topic modeling method summarized by Qiang et al. [18]. Within Wen et al.'s [11] approach, we suggest alternate strategies for selecting a suitable number of topics and an alternate method to analyze the evolution of code review comments. An overview of the steps followed in our study is depicted in Figure 1. In the figure, the approach by Wen et al. [11] is depicted in blue, and our changes are in green.

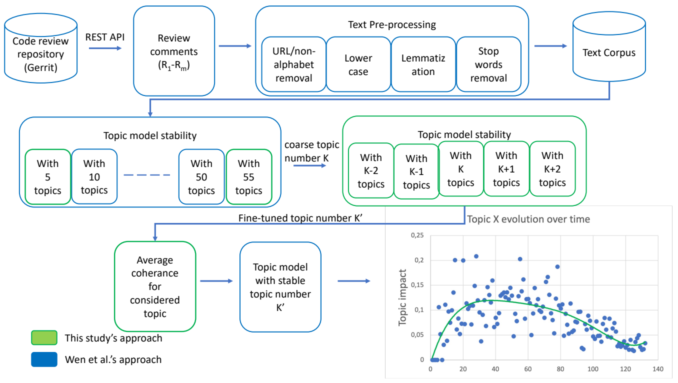

Fig. 1. Data extraction, pre-processing and topic modeling process, based on Wen et al. [11]

A. Datasets

We utilized code review comments from three open-source software (OSS), two OpenStack projects, Nova and Neutron, and LibreOffice. We included the same open-source system, Nova, as Wen et al. [11] to compare the performance of the short-text modeling method and LDA. The OpenStack community develops various storage and networking solutions and has been previously investigated in the studies on the code review process [19, 20]. The code review process of LibreOffice[2], an open-source office suite, also has been studied in the literature by several researchers [21, 22].

We used the REST API[3] provided by Gerrit to extract

[2]https://www.libreoffice.org/discover/libreoffice/

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

the code review comments from all three OSS. To study the evolution of code review themes over an extensive period, we extracted the code review comments between September 2011 till February 2023. We also only considered code review comments with more than two words.

B. Natural language processing model selection

One of the natural choices for the unsupervised classification of text data isLDA method [23]. LDA assumes that each document consists of a set of latent topics, whereas each latent topic consists of a group of words. LDA is shown to have degraded performance for short text [24] due to reduced word co-occurrence in short texts for topic extraction. Code review comments are relatively short pieces of text [25]. Therefore, we considered topic modeling models suitable for short text, as suggested by Qiang et al. [18].

Qiang et al. [18] compared several short-text topic modeling (STTM) methods and traditional LDA on six datasets. They found that STTM outperforms LDA in all six cases regarding classification accuracy and purity. In particular, Dirichlet multinomial mixture (DMM) based models outperformed global word co-occurrence-based short-text topic models and self-aggregation-based short-text topic models. We selected the Gibbs Sampling-based Dirichlet Multinomial Mixture model (GSDMM) [26] from among the DMM-based models due to its comparable classification accuracy and superior execution performance [18].

C. Data preprocessing

We replicate the pre-processing steps taken by Wen et al. [11] as depicted in Figure 1. After data extraction, we removed responses from the authors of the code commit to focusing only on code review comments from the reviewers. We also removed all brackets and punctuation to clean the code review comments and converted all code review comments to lowercase. We further removed URLs and words containing numbers as they did not contribute to the classification of themes in the code review comments. Next, we removed stop words using the built-in preprocessing library in the Gensim natural language processing toolkit. We also lemmatized all code review comments and removed any null strings in the code review comments.

TABLE I
OVERVIEW OF CODE REVIEW COMMENTS IN EXAMPLE SYSTEMS.

OSS Name	**Total code review comments**	**Average comment length (in words)**	**Stable number of topics**
Nova	102,642	28.2	24
Neutron	78,196	24.3	4
LibreOffice	28,328	26.8	11
Total	209,166	26.4	-

D. Parameter selection

The topic modeling algorithm's hyperparameters, such as the topic-to-document probability (*alpha*), word-to-topic probability (*beta*), and the number of topics (*N*), impact the performance of topic classification. A larger value for *N* may lead to fragmentation of topics, while a smaller value may tangle topics, thus compromising the semantic meaning of generated topics. A lower value for (*alpha*) limits the model to fewer topics per document, while higher (*beta*) results in a higher number of terms per topic. Generally, lower hyperparameter values lead to a more decisive model [27].

As suggested by Agarwal et al. [16] and used by Wen et al. [11], we use topic stability as a measure to identify suitable values for *N* for each corpus independently. Topic stability, a modified measure of cross-run similarity of topics based on Jaccard similarity [16], is the median number of word-terms occurrences in all considered runs for a given topic number. Extending the possible values of *N* used by Wen et al. [11], we considered *N*=[5..55] (in steps of five) for the number of topics to analyze topic stability and topic coherence for each of our datasets. We trained the GSDMM models for five runs, with the selected corpus sorted in a different order and varied choices for hyperparameters, *alpha*, and *beta*. We used the 10 top words from five runs for the GSDMM model to calculate topic stability for each dataset and identified the most stable number of topics. While Wen et al. [11] only evaluated a suitable choice of *N* in steps of five, we propose a two-stage approach for selecting a suitable *N*. As a first step, we select the most stable choice of *N* in steps of five (as did Wen et al.); we then iterate in steps of one in the neighborhood to find a more stable value for *N*.

In addition to using topic stability as suggested by Wen et al. [11], we also evaluate average coherence for all considered values of *N*. Using topic stability and average coherence value to select a suitable number of topics, *N*, in the previous step, we generate topics from the corpus and store the topic membership probabilities for each code review comment in the corpus. To compare the topic stability of GSDMM with LDA, we also repeated the above process for the LDA model.

E. Topic naming

Several methods have been utilized in the literature to assign an appropriate name to a topic. Silva et al. [17] classified these approaches as manual, automated, and a combination of manual and automated procedures. The manual naming approach has been frequently used in software engineering [11, 28, 29]. To identify a topic name, two authors read the top 20 words associated with a topic and the 20 unprocessed code review comments that have the highest topic membership to the topic. We read the code review comments belonging to each topic and assigned a label that captured the central theme within each code review comment. Then we reviewed all the labels belonging to a topic and assigned an appropriate name that captured most of the themes within the topic.

F. Topic impact evolution

Like Wen et al. [11], we study a given topic's evolution by plotting the topic's impact for each month, along with a five-degree polynomial-based regression line. A topic's impact is defined as the proportion of code review comments belonging to a specific topic within a month [30]. As suggested by Wen et al. [11], we used a low cut-off score for topic membership and included only code review comments with a topic membership probability ≥ 0.1 when calculating a given topic's impact.

In addition to Wen et al. [11], we also consider top code review comments belonging to months where the topic impact for a given topic has an interesting pattern. This may provide insight into how themes change over time and whether their sub-themes can be analyzed.

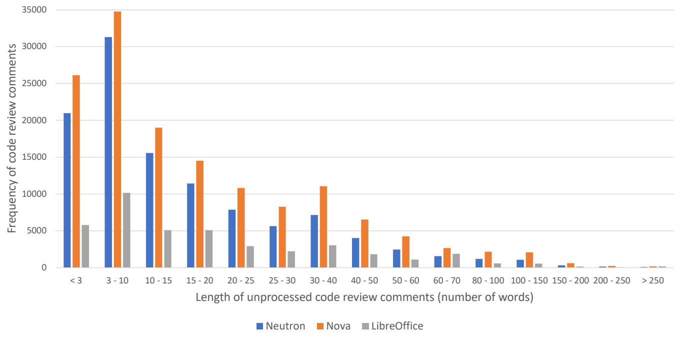

Fig. 2. Distribution of code review comments by length for the three example systems

IV. Results and analysis

After preprocessing the data as described in Section III-C, we got 209,166 code review comments from three OSS after removing approximately 84K code review comments due to short length. Their details are depicted in Table I. The distribution of code review comments by length for the three OSS is depicted in Figure 2. The evolution of the topic impact for all generated topics is provided in the replication package[4]. Here we first discuss our findings regarding the possible improvement areas. We also discuss themes for the system Neutron as a demonstration of the modified approach.

A. Improvement suggestions

1) Two-stage vs single-stage selection of N: As described in Section III-D, we utilized a two-stage process for selecting a suitable N, improving the average topic stability for the considered systems. Figure 3 depicts the average topic stability for the most stable choice of the number of topics for the single and two-stage topic selection. We observe that LibreOffice shows the highest improvement when using the two-stage topic selection, with average topic stability improving from 0.82 to 0.86.

2) Topic modeling method: To evaluate the variation in topic stability for the three systems for different values of N, we plot the average topic stability for all five runs for both GSDMM and LDA in Figure 4. Apart from system Neutron for N=4, GSDMM produces substantially more stable topics when compared to LDA for the considered systems. GSDMM, on average, produces 38% more stable topics than LDA for

[4]https://doi.org/10.5281/zenodo.7836738

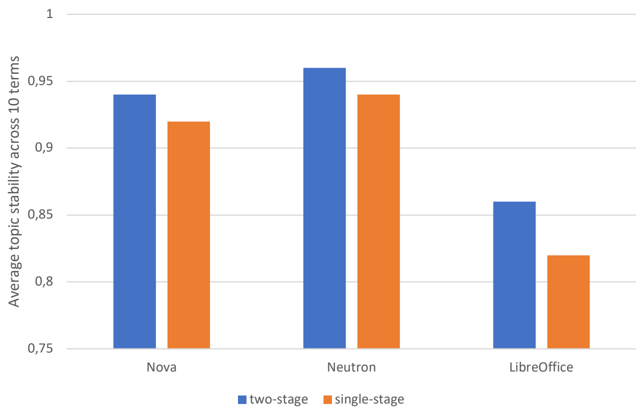

Fig. 3. Average topic stability for top 10 terms for two-stage (blue) and single-stage (orange) topic selection

system Nova. For the system Neutron, the average topic stability improvement is 32%. LibreOffice showed a similar pattern apart from the topic stability ratio for N=5. Based on the empirical results, short-text topic models such as GSDMM may be further evaluated when studying the evolution of code review comments.

We also observe that the most stable number of topics, N, may be less than 10, as in the case of Neutron, regardless of the topic modeling method used. There might be systems with most stable topic beyond N=55, thus plotting average topic stability for such systems may aid in selecting the upper and lower bounds for N.

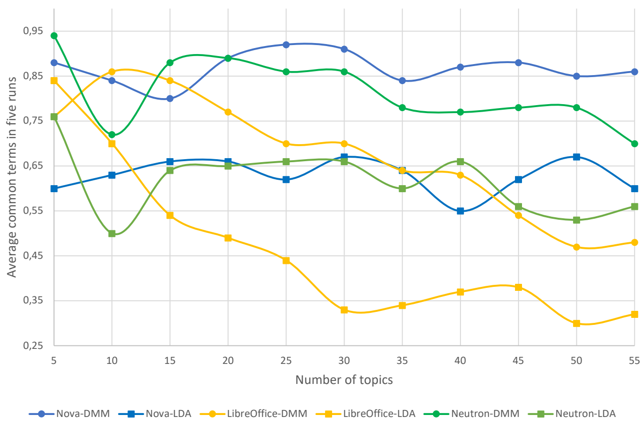

Fig. 4. Topic stability comparison using GSDMM and LDA for the three example systems

3) Topic stability and coherence: We also evaluated the average coherence of the topics for different values of N for all five runs. The results for both LDA and GSDMM for the three example systems are depicted in Figure 5. Compared to GSDMM, LDA achieves a slightly higher average coherence for all considered systems. The average coherence gradually increases with N for the considered systems and both algorithms, except for LibreOffice, where the average coherence declines after a peak around N=25. The highest average coherence for Neutron and Nova is 0.60 and 0.63, respectively, for N=55. In comparison, the trend for average topic stability is more varied for the increase in N. The contrasting results need further investigation.

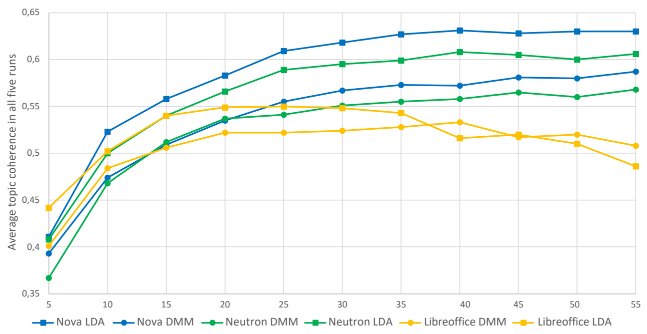

Fig. 5. Average coherence graph using LDA and GSDMM for the three example systems

The average coherence for the most stable number of topics is only slightly less than the most coherent topic for all considered systems. We preferred that the minor decrease in average coherence is an acceptable trade-off in selecting a suitable choice of N to study the evolution of the prevalent themes in code review comments.

4) Evolution analysis: one vs multiple windows: We hypothesize that some themes observed in code review comments may become obsolete with time as systems, processes, and methods to maintain the code base evolve. In contrast, new themes may replace them as code reviewers focus on current concerns within the submitted code. This raises the question regarding the timeframe to select for the analysis of themes. Choosing a very long duration may make it challenging to find relevant representative themes for all months considered. Similar to the approach by Wen et al. [11], we used a single window for the duration considered. In contrast, multiple windows approach divides the entire duration into sub-parts for a more refined analysis of the themes in code review comments.

As an exploration of the multiple windows approach, we analyzed the theme related to "guidelines" given its interesting evolution (see Figure 8). After generating the overall theme, we analyzed the top 20 code review comments from months when the trend changed significantly, along with code review comments from beginning to end, to evaluate if the issues discussed in the code review comments belonging to the topic evolved. We considered code review comments from November 2011, February 2020, and January 2023.

In the initial months, code review comments related to the code review process appear more frequently in the guidelines theme (e.g., "*...I think this would usually be a separate commit...*"). In February 2020, the guidelines theme focused more on code styling suggestions (e.g., "*L.28-38 belongs to L.26 so it looks better to indent these lines.*") while in the ending month, the guidelines theme considered code review style-related suggestions to be more critical (e.g., "*The patch is ok please remove the second line in the commit message.*"). One possible explanation for this trend is that guidelines provided by Neutron's core reviewers evolved with the change in contributing developer behavior.

Considering the pace of development technologies, processes, and practices improvements, we suggest generating topics in shorter windows, e.g., four months, rather than the entire duration. Using a "windowed" approach may refine the quality of themes identified as we can assess the impact of newly introduced interventions, e.g., company-wide procedural changes, in code review comments.

B. Named topics

In this section, we discuss the themes identified after generating topics from the Neutron system using the fine-tuned topic choice discussed above. The plots for topic evolution from the other two systems are provided in the replication package. The depicted scatter plots show the topic impact (see Section III-F) over the months considered. We also draw a polynomial-based trend line to represent the overall evolution of the topic impact across months. For Neutron, if all four topics were equally divided, we would expect an average topic impact of around 0.25 for each topic. We represent this with a continuous horizontal line at 0.25 in the generated plots.

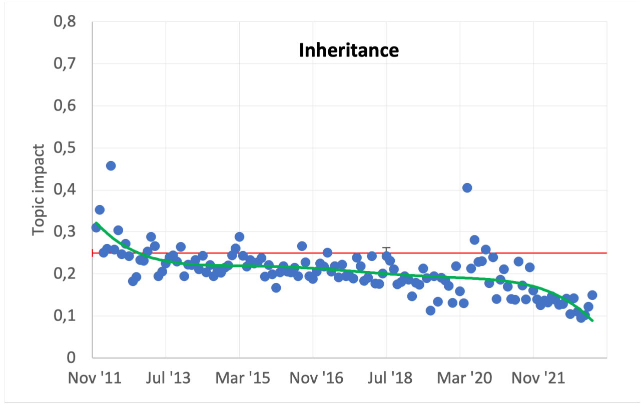

Fig. 6. Topic evolution for theme Inheritance in system Neutron.

1) Theme 1: Inheritance: We define inheritance-related themes as code review comments that discuss how classes inherit other classes, shared attributes and methods in multiple inheritance instances, and misuse of inheritance, among others. The ratio of code review comments belonging to the inheritance-related theme or topic share [30] is 0.33. The regression line shows that while the reviewers discussed inheritance-related issues more frequently in the initial months, the theme evolved with a gradual decrease in the topic impact over the duration considered. We reproduce an example of a code review comment discussing an inheritance-related theme.

"Althrough it's done everywhere is this class it's not pythonic to use the following (java?) pattern: *class MyClass: @staticmethod def method1(): pass @staticmethod def method2(): MyClass.method1()* because it breaks python inheritance principle: if a subclass *SubClass* of *MyClass* overloads method1 *SubClass* method2 will still use *MyClass.method1*! ..." (Italics added to improve readability)

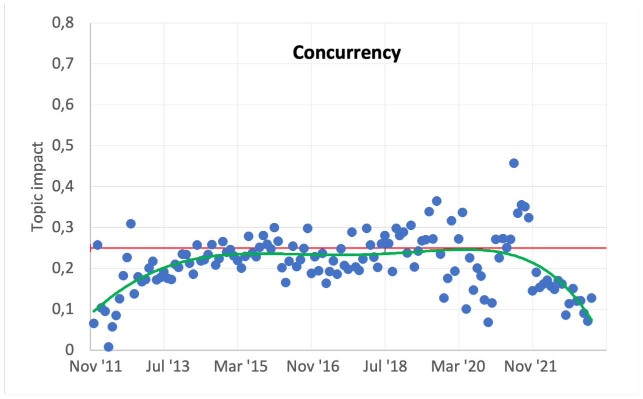

Fig. 7. Topic evolution for theme concurrency in system Neutron

2) Theme 2: Concurrency: We define concurrency-related themes as code review comments that discuss timing issues in a multi-threaded application, event handling, and network performance management. The topic share for concurrency-related code review comments is 0.39, making it the most prominent theme for Neutron. The evolution of this theme shows that topic impact has been steady between March 2015 till March 2020, after which it gradually declined to the levels observed before this period. An example of a code review comment discussing a concurrency theme is reproduced as follows.

"Not sure this is a good idea... what happens if another thread creates a new floating IP on this router between the time that another thread called this function and we get here? ..."

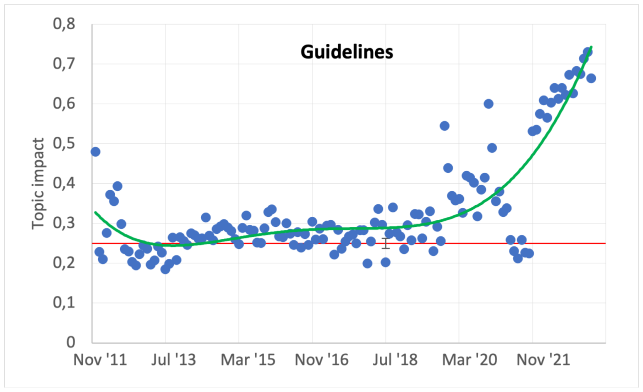

Fig. 8. Topic evolution for theme guidelines in system Neutron

3) Theme 3: Guidelines: We define the guidelines theme as code review comments focusing on issues related to formatting code commit messages, patch-related procedural guidelines, code styling suggestions, recommendations of what should be included in a commit message for clarity, and explaining why it was required. We observe that the guidelines theme undergoes a significant change during its evolution over the months considered. While the code review comments related to the theme remained steady between July 2013 till March 2020, there is a steady increase in code review comments related to guidelines. It would be interesting to analyze further the reasons for the sharp increase in the guidelines theme, which may lead to process-level improvement in how code reviews are performed in the Neutron community.

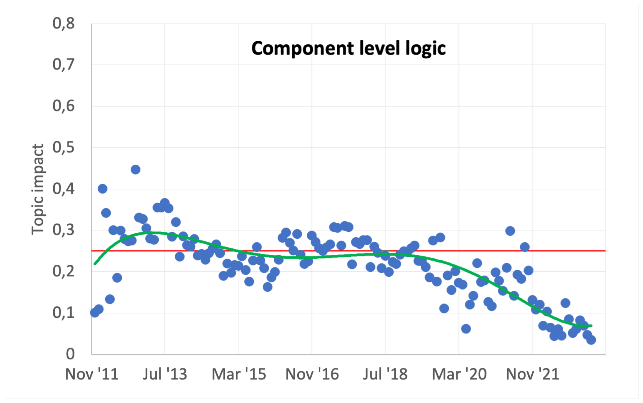

Fig. 9. Topic evolution for theme component level logic in system Neutron

4) Theme 4: Component level logic: We define component-level logic themes as code review comments discussing logic-related issues at the method or class level, such as conditional logic, suggestions for additional logic, tips to replace loops with alternate implementation, and recommendations for introducing alternate parameters in the implemented method. The percentage of code review comments belonging to this theme is only 17% making it the least discussed theme in Neutron. We observe that the topic impact for component-level logic theme remains steady till July 2018, after which it gradually declines below 0.1. We reproduce the following code review comment belonging to the theme.

"You can fold lines 130–133 into the for loop as *ichain_name* an *ochain_name* and then remove 137 138 141 142 So pull those 8 lines and add these two after *if* clause ending on line 142. ..." (Italics added to improve readability)

V. Discussion

We suggested three potential improvements to the approach of Wen et al. [11]. The two-stage selection of a suitable number for N improves topic stability compared to the single-stage approach. GSDMM produces substantially more stable topics as compared to LDA. Similarly, using average coherence in addition to topic stability provides a comparison between the two measures. Using the modified approach, we were able to identify several themes.

Themes similar to those we identified using our modified approach have also been discussed in the literature. The theme *component-level logic* is synonymous with *Logic* issues identified by Mäntyla et al. [6] and *code_logic* by Ochodek et al. [8]. The theme *concurrency* bears similarity with *Timing* issues identified in Mäntyla et al. [6]; however, we classify event handling and network-related performance in *concurrency*. The *guidelines* theme includes *code_style* issues from

Ochodek et al. [8], *visual representation* issues [6], as well as *code review process* [11]. The theme *inheritance* covers similar concepts as *structure* [6] and *code_design* [8] though it takes a fine-grained view of design-related code review comments. Intuitively, the topic share [30] for the themes can vary across the systems studied in the previous studies.

We achieved a different number of N for topic stability using the modified approach compared to Wen et al. [11] for Nova, possibly due to the difference in the duration considered and the topic modeling method used. Our modified approach produced a 60% more average topic stability than the state-of-the-art for Nova; however, we could not compare the topic stability for other systems considered. We also did not perform a comparison between the themes identified.

Several Dirichlet Multinomial Mixture (DMM) based topic modeling models have been reported in the literature. GSDMM assumes only one topic for each code review comment, which we consider a valid assumption given the short length of code review comments, especially for in-line code review comments. We selected GSDMM over the generalized Pòlya urn Poisson-based Dirichlet Multinomial Mixture model (GPU-PDMM), which had better classification accuracy than GSDMM in the four out of six datasets considered [18]. However, we selected GSDMM for its fast execution time and slightly lower classification accuracy than GPU-PDMM. GPU-PDMM may improve topic stability and average coherence in similar studies investigating topic evolution in code review comments. Similarly, several studies have proposed bi-directional transformer-based (BERT) topic modeling methods [31, 32]; however, a qualitative comparison between the performance of BERT and STTM on code review comments may improve the evolution analysis of code review comments.

There can be different approaches for using multiple windows to analyze the evolution of themes in code review comments. We only demonstrated analyzing selective months based on the trend. A sliding window approach can also be used to analyze the evolution of themes for code review comments in a given set of months; then, we update the window by adding newer months while removing the oldest months. However, further research is needed to qualitatively evaluate these approaches for the quality of the generated themes.

The identified themes and an analysis of how the themes with high topic impact evolve can lead to crucial changes in how teams approach development tasks, company development guidelines, and process-level improvements. As an illustration, an analysis of the *guidelines* theme may provide a checklist for developers to self-check before submitting source code. As the theme evolves, the checklist may be updated and thus remains relevant for the contributing developers. The updated analysis provides an initial step toward developing data-driven dashboards for practitioners to aid in the study of important themes which can be utilized to suggest improvement directions.

We have demonstrated that incorporating the suggestions leads to quantitative improvements in topic stability. However, we have yet to evaluate if these design suggestions lead to qualitative improvement in the quality of the themes identified.

Wen et al. [11] demonstrated that their approach was suitable for analyzing one closed-source system. Our results indicate that the updated approach may also be well-suited for analyzing closed-source systems. The effectiveness of the analysis of common themes and their evolution is intuitively dependent on the quality of the code review comments. Previous studies have observed that experienced reviewers with in-depth knowledge of the project provide context-specific feedback that may lead to more meaningful common themes using our approach. As a future study, we intend to use the updated approach and interview code reviewers regarding the quality of the themes produced.

VI. Threats to validity

A. Data validity

We utilized the three open-source datasets and used random sampling from the dataset to evaluate the quality of the individual dataset. We removed the code review comments with an entry in "in reply to" to remove discussion replies from developers. We used an embedding model [33] designed from posts in StackOverflow [5], a platform to discuss software code issues, to further improve the quality of the generated topic from short-text modeling methods. While an embedding model designed from code review comments may improve results, we believe our selected word embedding model captures essential information related to software development and is a suitable option. Only one author was involved in the data extraction and topic stability evaluation. However, we used automated tools and scripts where possible in these stages to keep the possibility of human error to a minimum. We also did not consider removing highly frequent words during preprocessing, which may impact the results presented.

B. Research validity

To improve the repeatability of our study, we have shared the datasets and Python scripts online as part of our replication package. We have also described our steps in preprocessing data, and the topic selection process. Moreover, to reduce the chances of researcher bias, two authors were involved in assigning names to generated topics.

C. External validity

The empirical study presented may have a few threats relating to its external validity and limit the generalizability of the results. Since we selected only open-source systems, the language used in the code review comments may vary for other open-source and industrial systems. Intuitively, the language can inherently differ from one reviewer to another; thus, the number of reviewers involved in the review also impacts the language in code review comments. Further studies are needed using varied datasets from both open-source and industry to assess the generalizability of our approach.

[5]https://stackoverflow.com/

VII. Conclusions

Building on the recent study by Wen et al. [11], we performed an exploratory study to evaluate possible design improvements in the study of the evolution of common themes in code review comments. Among other design and analysis improvements, we observed that the short-text modeling method leads to more stable topics than traditional topic modeling. Studying the evolution of common themes in code review comments is a promising field, with practical implications for research and practice that may lead to suggestions that help improve the development and process-related practices. By extending their work and proposing new approaches for topic selection and analysis of topic evolution, we highlight that the choice of modeling technique is essential as it may lead to different results. Further studies are needed in code review comments evolution and analysis to investigate the suggestions made. In future work, we aim to use industrial datasets along with interviews with reviewers to investigate the reasons behind the changes in the interesting themes as well as their reflections on using the identified themes to create data-driven dashboards and interventions at the development or process level that may aid in improving the issues highlighted in the derived themes.

Acknowledgments: This work has been supported by ELLIIT, the Strategic Research Area within IT and Mobile Communications, funded by the Swedish Government.

References

[1] S. McIntosh, Y. Kamei, B. Adams, 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, in: Proceedings of the 11th working conference on mining software repositories, 2014, pp. 192–201.

[2] G. Bavota, B. Russo, Four eyes are better than two: On the impact of code reviews on software quality, 2015.

[3] S. McConnell, Code complete, Pearson Education, 2004.

[4] A. Bacchelli, C. Bird, Expectations, outcomes, and challenges of modern code review, in: Proceedings of the 35th International Conference on Software Engineering, 2013, pp. 712–721.

[5] A. Bosu, J. C. Carver, C. Bird, J. Orbeck, C. Chockley, Process aspects and social dynamics of contemporary code review: Insights from open source development and industrial practice at microsoft, IEEE Transactions on Software Engineering 43 (2017) 56–75.

[6] M. V. Mäntylä, C. Lassenius, What types of defects are really discovered in code reviews?, IEEE Transactions on Software Engineering 35 (2009) 430–448.

[7] M. Beller, A. Bacchelli, A. Zaidman, E. Juergens, Modern code reviews in open-source projects: which problems do they fix?, in: Proceedings of the 11th Working Conference on Mining Software Repositories, 2014, pp. 202–211.

[8] M. Ochodek, M. Staron, W. Meding, O. Söder, Automated code review comment classification to improve modern code reviews, in: Proceedings of the 14th International Conference on Software Quality, 2022, pp. 23–40.

[9] R. Chatley, L. Jones, Diggit: Automated code review via software repository mining, in: Proceedings of the 25th International Conference on Software Analysis, Evolution and Reengineering, 2018, pp. 567–571.

[10] R. Tufano, L. Pascarella, M. Tufano, D. Poshyvanyk, G. Bavota, Towards automating code review activities, in: Proceedings of the 43rd International Conference on Software Engineering, 2021, pp. 163–174.

[11] R. Wen, M. Lamothe, S. McIntosh, How does code reviewing feedback evolve?: A longitudinal study at Dell EMC, in: Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice, 2022, pp. 151–160.

[12] S. Gunawardena, E. Tempero, K. Blincoe, Concerns identified in code review: A fine-grained, faceted classification, Information and Software Technology 153 (2023) 107054.

[13] Y. Hong, C. Tantithamthavorn, P. Thongtanunam, A. Aleti, CommentFinder: A simpler, faster, more accurate code review comments recommendation, in: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2022, pp. 507–519.

[14] F. Zampetti, S. Mudbhari, V. Arnaoudova, M. Di Penta, S. Panichella, G. Antoniol, Using code reviews to automatically configure static analysis tools, Empirical Software Engineering 27 (2022) 28.

[15] Y. Arafat, S. S. H. Shamma, Categorizing review comments by mining software repositories, International Conference on Advances in Computing and Data Sciences (2020) 12.

[16] A. Agrawal, W. Fu, T. Menzies, What is wrong with topic modeling? And how to fix it using search-based software engineering, Information and Software Technology 98 (2018) 74–88.

[17] C. C. Silva, M. Galster, F. Gilson, Topic modeling in software engineering research, Empirical Software Engineering 26 (2021) 1–62.

[18] J. Qiang, Z. Qian, Y. Li, Y. Yuan, X. Wu, Short text topic modeling techniques, applications, and performance: a survey, IEEE Transactions on Knowledge and Data Engineering 34 (2020) 1427–1445.

[19] P. Thongtanunam, S. McIntosh, A. E. Hassan, H. Iida, Revisiting code ownership and its relationship with software quality in the scope of modern code review, in: Proceedings of the 38th International Conference on Software Engineering, 2016, pp. 1039–1050.

[20] D. M. German, G. Robles, G. Poo-Caamaño, X. Yang, H. Iida, K. Inoue, "was my contribution fairly reviewed?" A framework to study the perception of fairness in modern code reviews, in: Proceedings of the 40th International Conference on Software Engineering, 2018, pp. 523–534.

[21] G. Bavota, B. Russo, Four eyes are better than two: On the impact of code reviews on software quality, in: Proceedings of the 31st International Conference on Software Maintenance and Evolution, 2015, pp. 81–90.

[22] X. Xia, D. Lo, X. Wang, X. Yang, Who should review this change?: Putting text and file location analyses together for more accurate recommendations, in: Proceedings of the 31st International Conference on Software Maintenance and Evolution, 2015, pp. 261–270.

[23] D. M. Blei, A. Y. Ng, M. I. Jordan, Latent dirichlet allocation, Journal of Machine Learning research 3 (2003) 993–1022.

[24] X. Cheng, X. Yan, Y. Lan, J. Guo, BTM: Topic modeling over short texts, IEEE Transactions on Knowledge and Data Engineering 26 (2014) 2928–2941.

[25] Z. Li, Y. Yu, G. Yin, T. Wang, Q. Fan, H. Wang, Automatic classification of review comments in pull-based development model, in: Proceedings of the 29th International Conference on Software Engineering and Knowledge Engineering, 2017, pp. 572–577.

[26] J. Yin, J. Wang, A dirichlet multinomial mixture model-based approach for short text clustering, in: Proceedings of the 20th International Conference on Knowledge Discovery and Data Mining, 2014, pp. 233–242.

[27] L. R. Biggers, C. Bocovich, R. Capshaw, B. P. Eddy, L. H. Etzkorn, N. A. Kraft, Configuring latent dirichlet allocation based feature location, Empirical Software Engineering 19 (2014) 465–500.

[28] M. U. Haque, L. H. Iwaya, M. A. Babar, Challenges in docker development: A large-scale study using stack overflow, in: Proceedings of the 14th International Symposium on Empirical Software Engineering and Measurement, 2020, pp. 1–11.

[29] J. Han, E. Shihab, Z. Wan, S. Deng, X. Xia, What do programmers discuss about deep learning frameworks, Empirical Software Engineering 25 (2020) 2694–2747.

[30] A. Barua, S. W. Thomas, A. E. Hassan, What are developers talking about? An analysis of topics and trends in stack overflow, Empirical Software Engineering 19 (2014) 619–654.

[31] R. Egger, J. Yu, A topic modeling comparison between LDA, NMF, top2vec, and BERTopic to demystify twitter posts, Frontiers in Sociology 7 (2022) 886498.

[32] M. Grootendorst, BERTopic: Neural topic modeling with a class-based TF-IDF procedure, arXiv preprint arXiv:2203.05794 (2020).

[33] V. Efstathiou, C. Chatzilenas, D. Spinellis, Word embeddings for the software engineering domain, in: Proceedings of the 15th International Conference on Mining Software Repositories, 2018, pp. 38–41.