Dynamic Evolution of Source Code Topics

Khaled Almustafa
College of Engineering
Prince Sultan University
Riyadh 11586, Saudi Arabia
Email: kalmustafa@psu.edu.sa

Mamdouh Alenezi
College of Computer and Information Sciences
Prince Sultan University
Riyadh 11586, Saudi Arabia
Email: malenezi@psu.edu.sa

Abstract—**Open-source projects continue to evolve that result in so many versions. Analyzing the unstructured information in the source code is based on the idea that the unstructured information reveals, to some extent, the concepts of the problem domain of the software. This information adds a new layer of source code semantic information and captures the domain semantics of the software. Developers shift their focus on which topic they work more in each version. Topic models reveal topics from the corpus, which embody real world concepts by analyzing words that frequently co-occur. These topics have been found to be effective mechanisms for describing the major themes spanning a corpus. Previous Latent Dirichlet Allocation (LDA) based topic analysis tools can capture strengths evolution of various development topics over time or the content evolution of existing topics over time. Regrettably, none of the existing techniques can capture both strength and content evolution. In this work, we apply Dynamic Topic Models (DTM) to analyze the source code over a period of 10 different versions to capture both strength and content evolution simultaneously. We evaluate our approach by conducting a case study on a well-known open source software system, jEdit. The results show that our approach could capture not only how the strengths of various development topics change over time, but also how the content of each topic (i.e., words that form the topic) changes over time which shows that our approach can provide a more complete and valuable view of software evolution.**

Keywords–Open source; Source code; LDA; Topic extraction; Software evolution.

I. Introduction

Program comprehension is an essential activity in the course of software maintenance and evolution [1]. Typically, developers spend around 60% of their working hours comprehending the system while doing software maintenance tasks [1], especially the source code. Monitoring, visualizing and understanding the evolution of a large system are essentially challenging tasks.

Comprehending how source code topics evolve over time can be a great help for project stakeholders and managers to observe and understand activities and efforts performed on a software repositories over time. For instance, project managers can observe what feature the development team is working on by consulting the source code repository and developers can observe a specific feature evolution by consulting the same source code repository as well [2]–[4].

Several LDA-based techniques were proposed with the aim of supporting software projects stakeholders, managers, and developers to comprehend software evolution. Thomas et al. [4] used the Hall model [5] to study the history of source code to find out the strengths (i.e., popularity) of the change of several topics over time. They applied LDA one time on all versions of a specific software project to extract the topics and computed different metrics to show the strength of each topic for each version. Their approach can capture the development evolution strength. However, the content of a topic (i.e., the set of words that form a topic), never changes across the versions. Differently, Hindle et al. [6] used the Link model [7] that runs a separate LDA for each time window and used a different step to link similar topics. Their approach can capture changes in the content of each topic over time (i.e., content evolution). Unfortunately, their approach was not able to recover the strength of a topic across all time windows because of the lack of available information (time windows). Consequently, none of the current approaches can capture both strength and content evolution.

As we have seen, current approaches on understanding source code topics evolution emphasized on the strength of the evolution or the content evolution. However, both strength and content of the evolution are essential for developers to entirely comprehend how software evolves. For instance, project managers want to figure out how much effort is dedicated to feature X at a specific time, which can be attained by calculating topic strength. They may also want to figure out what kind of effort was done on feature X at a specific time point. By itself, topic strength will not assist project managers with this kind of information. Instead, the content of a specific topic can give some insights and shed some light on activities which performed on feature X at a specific point of time.

In this work, we propose a new approach to capture source code evolution from two different dimensions: strength and content. We apply Dynamic Topic Models (DTM) [8] on the source code of a software repository. After that, we capture topic strength evolution by calculating the Normalized-Assignment metric at each software release to represent the strength of a topic for that time. We capture topic content evolution by extracting the top 10 words that characterize a topic for each software release.

We conduct an empirical study on the source code of a well-known open source software system, jEdit. The results show that the new approach can capture both strength and contents of different source code topics over time, which corresponds to meaningful description of the whole development iteration, hence, a more complete view of software evolution.

An essential step towards comprehending and understanding the functional behavior of any system is to find and recognize business topics that exist in the source code of

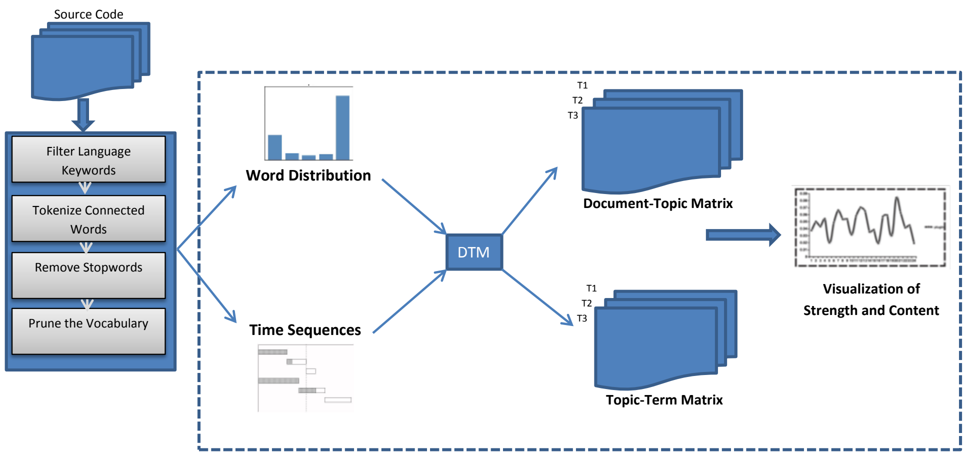


Figure 1. The Proposed Approach

the system. These business domain objects are modeled as high level components and then realized in the implementation and transformed into code. For example consider an UML modeling application that models UML diagrams and deals with objects, figures, relationships, and cardinality. When a maintainer with no application knowledge wants to add a new feature or modify one of the features, he/she will find it very difficult before comprehending and understanding the main functionality of the application. Extracting business topics from the source code and establishing the relationship between them would be a huge support in finding related data structures, methods, classes. This will eventually help the developer or the maintainer productivity, especially when dealing with a large system with little documentation.

The rest of the paper is organized as follows: Section II discusses topic evolution models in more depth. Section III discusses the approach used in this work. Section IV presents the experimental evaluation and discussions. Section V discusses threats to the validity of the study. Section VI discusses some related work to this work. Section VII concludes the paper.

II. Topic Evolution Models

Topic evolution model is taking into consideration the time while modeling the topics. It models how certain topics evolve and change over time. For a specific topic, the strength of that topic can change several times over time (spikes and drops) during the lifetime of a corpus. Furthermore, a specific topic content may likewise change over time, designating different aspects of the evolution within a specific topic. As we have seen in the introduction section, numerous topic evolution models were proposed in the literature.

1) The Link Model, which was proposed by Mei et al. [7]: Hindle et al. [6] were the first ones who applied this approach to analyze the topic evolution of commit messages. They utilized topic modeling (LDA) separately for each time. Then they used a post-processing phase to link similar topics across successive time interval.
2) The Hall model, which was proposed by Hall et al. [5]: Linstead et al. [9] were the first ones who applied this approach to analyze source code evolution. The same approach was used and validated by Thomas et al. [10] to model source code changes. The Hall model applies LDA to the whole versions of the software corpus (all releases included in the study). Then a post-processing phase is used to separate corpus at different versions and different metrics are calculated to represent how much the contribution of this topic in a specific version.
3) Dynamic Topic Models (DTM), which was proposed by Blei et al. [8]: DTM is what is used in this work. It models a topic evolution as a discrete Markov process with normally distributed changes between time periods that allows only steady changes over time. It uses time-sequentially organized documents of the corpus to capture the topics evolution and creates a document-topic matrix and topic-term matrix at each time period. Document-topic matrix represents each document as multi-membership mixture of topics to show the topic strength evolution. Topic-term matrix represents each topic as a multi-membership mixture of terms to show the topic content evolution

III. The Proposed Approach

Figure 1 shows a high-level overview of our approach. We first obtain the source code of the studied systems. Second, we filter out noisy data by applying a number of preprocessing steps on the corpus. Third, we determine the optimal number of topic for this corpus. Fourth, we obtain word distributions and the releases into time sequences. These distributions and sequences are used as input for DTM to produce document-topic matrix and topic-term matrix at each release. Our approach is very similar approach to Hu et al. [11] but it is different than their approach in two main aspects: the target of the topic modeling (commits vs source code) and choosing the number of topics (random vs well-established method).

A. Data Preprocessing

A number of pre-processing steps are applied to the source code [4]. These pre-processing steps are common in most information retrieval techniques [1]. First, syntax and programming language keywords are filtered out. Second, each word is then tokenized according to well-known naming practices, for instance, underscores (first_name) and camel case (firstName). Third, common English terms are removed (stop words) to eliminate noise. The final step is to prune the vocabulary. The number of terms that can end up the bag-of-words is very large, which usually would cause a problem in most text-mining applications. In order to select the most useful subset, a filter has been applied to remove the overly common terms that appear in too many documents (=90%), as they can be seen as a non-informative and background terms.

B. Choosing K

Choosing how many topics to use in topic models is still a research problem not only for the source code domain. Topic excerption in textual documents also encounters the same problem. In this work, we adopted a well-known method for determining the number of topics [12]. This method specifies that the number of topics K can be determined by running LDA for different values of K with freezing the LDA hyper-parameters. For each value of K, they estimate the symmetric Kullback-Leibler divergence [13] between the singular values of the topic-word matrix and the document-topic matrix using the following equation:

$$Measure = KL(CM1||CM2) + KL(CM2||CM1) \quad (1)$$

The method calculates the symmetric Kullback-Leiber divergenece of the Singular value distributions of of two matrices M1 and M2. In this equation, CM1 represents the singular values distribution of the topic word matrix, CM2 represents the distribution obtained by normalizing the vector L*M2 where L is a one-dimensional vector of documents lengths in the corpus and M2 is the document topic-matrix. To determine K, choose K where the minimum value of the measure is. It is noteworthy that these distributions CM1 and CM2 are in sorted order.

C. Running Dynamic Topic Models

After we filtered noisy words in the corpus by applying the pre-processing steps, we use DTM to generate the document-topic matrix and topic-term matrix for each release after we feed the DTM the word distribution and the time sequences. Dynamic topic modeling applies these steps using several sequential time slices in the data set. We wrote an R script that applies our approach. We used the 'topicmodel' package version 0.2-1 in the R language version 3.1.12. The LDA parameters were chosen based on the recommendation of the literature [14]. The used parameters are = 50/K and = 0.01 where K is the number of topics.

D. Visualization of Strength and Content

After applying our approach, we visualize the results using the document-topic matrix and topic-term matrix. We calculate several topic metrics to symbolize both the strength and content of topics evolutions. We measure how the topic strength changes over time by computing a normalized assignment metric at each time point. This metric is the average value of the topic memberships of all documents in that topic at a time, which indicates the total presence of the topic in that time. The strength evolution of a topic is a time-indexed vector of normalized assignment values for that topic. Then, for the topic content, which includes the words and their distributions, it contains two parts: the word and its frequency within a topic. We choose the top 10 most frequent words to illustrate a topic and measure how the topic content changes over time by computing Term-Frequency (TF) at each time point.

IV. Results

In this section, we applied our approach to the repository of jEdit. We show the example topics and the captured information of version 3 of jEdit in Table I. We discuss our results and the empirical data in detail. The results are demonstrated in Figure 2, Figure 3, and Figure 4. These figures display a number of top words from selected topics in each version based on the term frequency (TF) value of each word in that topic. By looking at the figures, we observe the topic strength evolution by mapping the assignment value of topics to versions as well as the topic content evolution by mapping TF values of several top words to versions.

We applied our approach to the repository of jEdit from 2002 to 2012, which includes 10 main releases (3.0, 3.1, 3.2, 4.0, 4.1, 4.2, 4.3, 4.4.1, 4.5, 5.0). We found that most topics' strength evolution fluctuated greatly during the studied time period, which indicates that development topics are varied and distributed in each version. Furthermore, we observed that the top two words across different topics are add and plugin, which represents the active growth of plugins, an essential feature of jEdit. We studied each topics strength and content. We found three important topics:

Selection. Topic 1 relates heavily to selections done in jEdit. We found that the most frequent words contained in this topic were line, text, selection, length and area. The topic's strength reaches to peak (8%) in version 9 (4.5), and before that time it varies from 3.1% to 5.8% and fluctuates greatly. We looked in more details at the top words in version 9, and found that the TF of these words seems to decline after version 9. This was the real content evolution trend of this topic, because we found that the top words rarely appeared in previous versions.

Formatting. In topic 5, words such as color, window, font, height and width are common throughout the whole selected versions, which clearly represent the continuing availability of these features in jEdit. The topics strength varies from 1.9% to 8% and also experiences great fluctuation. The peak strength was reached in version 2, but after other functionalities, which implies other development topics, the strength fell down in consecutive versions.

GUI. In topic 7, words such as border, handler, action, button, panel, area and event are common during the whole time, meaning that several components relate to GUI. The topic's strength reaches to peak (about 15%) in version 8 and we found that about 20% of the top matching words come from this version. We also found that border is a notable GUI feature provided by jEdit because the word border is almost the top frequent term within this topic from the beginning to end.

There are other development topics, such as bug fixing and feature requests, directory for plugins, buffer and bufferset, regular expressions, build system files and code cleanup.

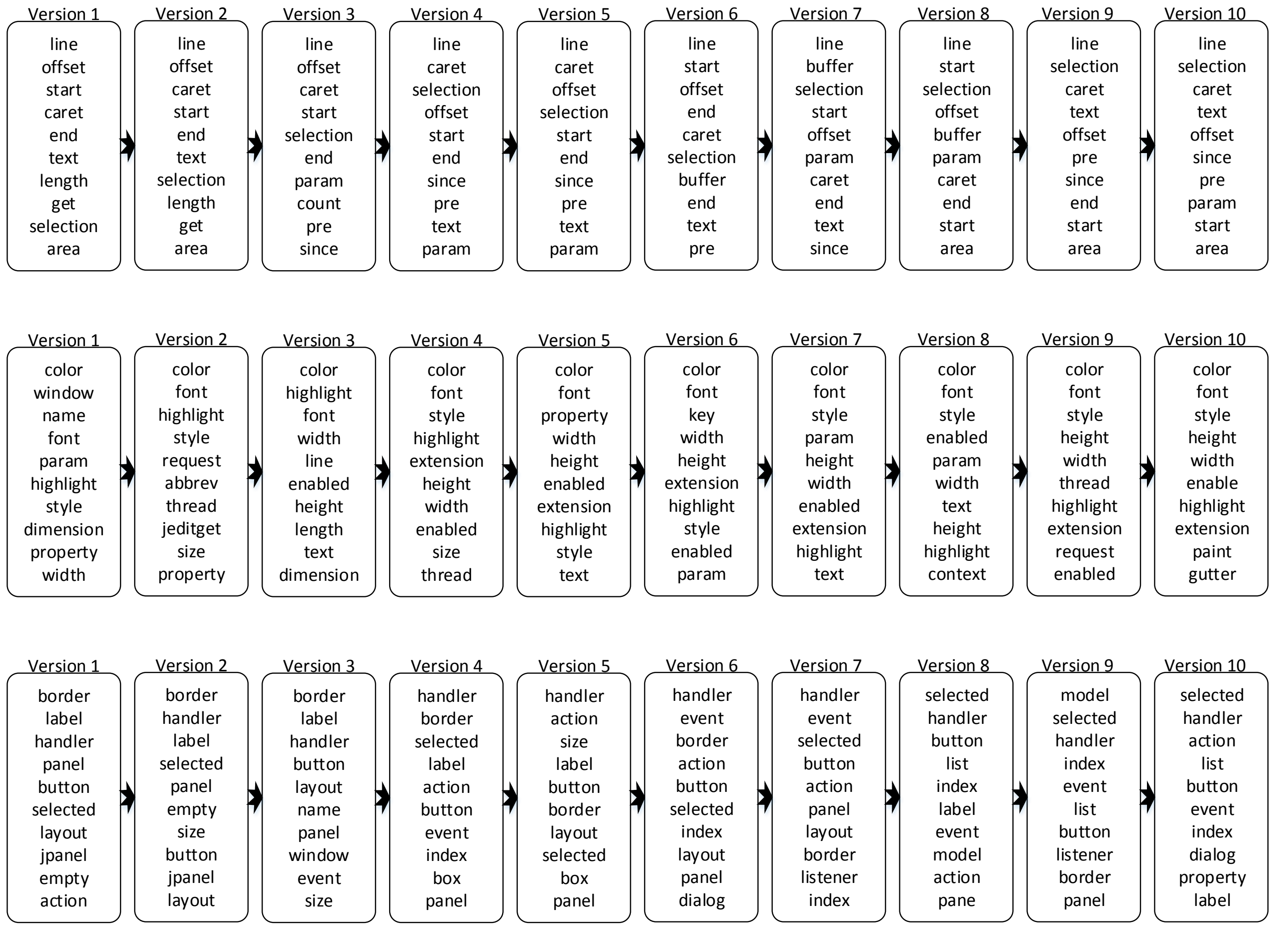

Version 1
line offset start caret end text length get selection area
Version 2
line offset caret start end text selection length get area
Version 3
line offset caret start selection end param count pre since
Version 4
line caret selection offset start end since pre text param
Version 5
line caret offset selection start end since pre text param
Version 6
line start offset end caret selection buffer end text pre
Version 7
line buffer selection start offset param caret end text since
Version 8
line start selection offset buffer param caret end start area
Version 9
line selection caret text offset pre since end start area
Version 10
line selection caret text offset since pre param start area
Version 1
color window name font param highlight style dimension property width
Version 2
color font highlight style request abbrev thread jeditget size property
Version 3
color highlight font width line enabled height length text dimension
Version 4
color font style highlight extension height width enabled size thread
Version 5
color font property width height enabled extension highlight style text
Version 6
color font key width height extension highlight style enabled param
Version 7
color font style param height width enabled extension highlight text
Version 8
color font style enabled param width text height highlight context
Version 9
color font style height width thread highlight extension request enabled
Version 10
color font style height width enable highlight extension paint gutter
Version 1
border label handler panel button selected layout jpanel empty action
Version 2
border handler label selected panel empty size button jpanel layout
Version 3
border label handler button layout name panel window event size
Version 4
handler border selected label action button event index box panel
Version 5
handler action size label button border layout selected box panel
Version 6
handler event border action button selected index layout panel dialog
Version 7
handler event selected button action panel layout border listener index
Version 8
selected handler button list index label event model action pane
Version 9
model selected handler index event list button listener border panel
Version 10
selected handler action list button event index dialog property label

Figure 2. 10 Top Words Evolution for Topics.

TABLE I. EXAMPLE TOPICS AND THE CAPTURED INFORMATION OF VERSION 3 OF JEDIT

Topic	Top 10 Words
Selection	line, offset, caret, start, selection, end, param, count, pre, since
Formatting	color, highlight, font, width, line, enabled, height, length, text, dimension
Menu	border, label, handler, button, layout, name, panel, window, event, size

V. THREATS TO VALIDITY

In this section, we discourse some limitations to our study. The approach of this work depends on the quality of comments and identifier names found in the code. jEdit is known for its robust designs, extensive documentation, strict coding and naming conventions. In addition, a previous study revealed that majority of java systems have good comments and good identifiers names, which make them sufficient for such topic analyses [15].

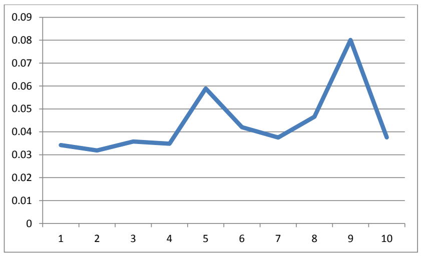
Regarding pre-processing steps, we performed four different steps on the source code. However, there is no consensus in the literature on which steps are essential or beneficial. Regarding parameter values, we used a well-established approach to find the optimal number of topics, which is much better than previous work in which they randomly selected a number of topics. We have focused on one open source Java-based systems. however, we cannot generalize the results. Additional case studies are needed to investigate closed-source and other programming languages systems.

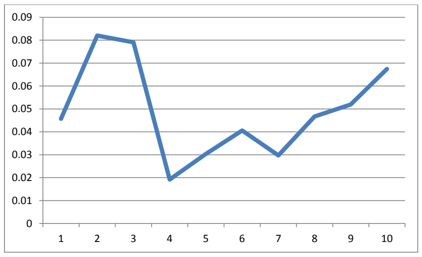
VI. RELATED WORK

Mining software repositories is booming in the software engineering research community these recent years [16]–[18]. We discuss some of the related work on mining software repositories efficiently to help software maintenance tasks.

Sun et al. [19] proposed an approach based on LDA to find out what kind of historical information is needed to support software maintenance. They evaluated their approach by a new study on another important software maintenance task, i.e., feature location. Furthermore, their benchmarked their studies with more subject programs and metrics.

Herzig et al. [21] conducted empirical studies of tangled changes, which introduce noise in software repositories [20]. Their results were promising in which they showed that about


(a) Selection


(b) Formatting

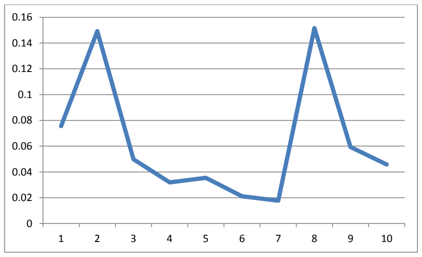
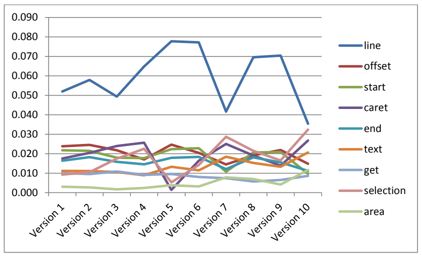
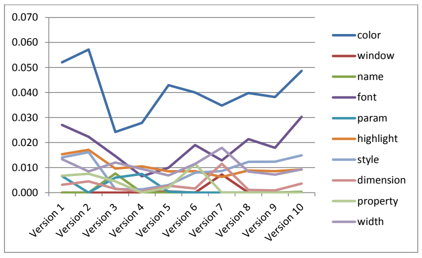

(c) GUI

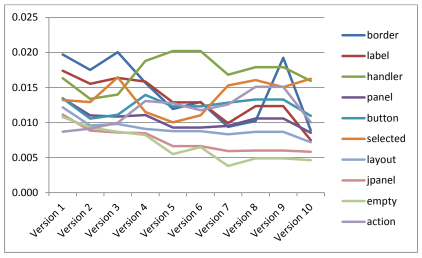
Figure 3. Topic Strength (Normalized Assignment).



(a) Selection



(b) Formatting



(c) GUI

Figure 4. Topic Content (TF values of top words within software versions).

20% of all bug fixes consist of multiple tangled changes. Keivanloo et al. proposed a collaborative platform for the purpose of sharing software datasets. Their platform supports data extraction, integration from various version control, issue tracking, and quality evaluation repositories. Their main focus was the integration of the information in software repositories.

Thomas et al. [4] proposed to use LDA to study the software evolution. They investigated whether the topics from the LDA corresponded well with actual code changes. Their results showed the effectiveness of using topic models as tools for studying the evolution of a software system. Furthermore, Their studies provided a good motivation for other researchers to use the topic model to mine the topics from software repositories.

VII. Conclusion

Available topic evolution models address only strength evolution or content evolution of the unstructured software repositories, not both like our approach. Having both the content evolution and the strength evolution will provide more comprehensive and complete results in order to understand the evolution of the source code than either one of them. In this work, we applied the Dynamic Topic Models to the source code to represent both their topic strength and content evolution. An empirical analysis of one well-known and open source projects, jEdit was conducted. We found that DTM produce complete and comprehensive view of software evolution, which is useful for a variety of stallholders to understand the changes of development topics from different aspects in a version. A future direction would be choosing more finest pre-processing steps. Another direction is to apply this approach on more software systems and conduct more comparative studies

Acknowledgment

The Authors would like to thank Prince Sultan University (PSU), Riyadh, K.S.A. for partially supporting this project.

References

[1] M. Alenezi, “Extracting high-level concepts from open-source systems,” International Journal of Software Engineering and Its Applications, vol. 9, no. 1, 2015, pp. 183–190.

[2] A. Panichella, B. Dit, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia, “How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms,” in Proceedings of the 2013 International Conference on Software Engineering. IEEE Press, 2013, pp. 522–531.

[3] L. R. Biggers, C. Bocovich, R. Capshaw, B. P. Eddy, L. H. Etzkorn, and N. A. Kraft, “Configuring latent dirichlet allocation based feature location,” Empirical Software Engineering, vol. 19, no. 3, 2014, pp. 465–500.

[4] S. W. Thomas, B. Adams, A. E. Hassan, and D. Blostein, “Studying software evolution using topic models,” Science of Computer Programming, vol. 80, 2014, pp. 457–479.

[5] D. Hall, D. Jurafsky, and C. D. Manning, “Studying the history of ideas using topic models,” in Proceedings of the conference on empirical methods in natural language processing. Association for Computational Linguistics, 2008, pp. 363–371.

[6] A. Hindle, M. W. Godfrey, and R. C. Holt, “What’s hot and what’s not: Windowed developer topic analysis,” in Software Maintenance, 2009. ICSM 2009. IEEE International Conference on. IEEE, 2009, pp. 339–348.

[7] Q. Mei and C. Zhai, “Discovering evolutionary theme patterns from text: an exploration of temporal text mining,” in Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining. ACM, 2005, pp. 198–207.

[8] D. M. Blei and J. D. Lafferty, “Dynamic topic models,” in Proceedings of the 23rd international conference on Machine learning. ACM, 2006, pp. 113–120.

[9] E. Linstead, C. Lopes, and P. Baldi, “An application of latent dirichlet allocation to analyzing software evolution,” in Seventh International Conference on Machine Learning and Applications, ICMLA 2008. IEEE, 2008, pp. 813–818.

[10] S. W. Thomas, B. Adams, A. E. Hassan, and D. Blostein, “Validating the use of topic models for software evolution,” in 10th IEEE Working Conference on Source Code Analysis and Manipulation (SCAM), 2010. IEEE, 2010, pp. 55–64.

[11] J. Hu, X. Sun, D. Lo, and B. Li, “Modeling the evolution of development topics using dynamic topic models,” in 22nd IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 2015, pp. 3–12.

[12] R. Arun, V. Suresh, C. V. Madhavan, and M. N. Murthy, “On finding the natural number of topics with latent dirichlet allocation: Some observations,” in Advances in Knowledge Discovery and Data Mining. Springer, 2010, pp. 391–402.

[13] T. M. Cover and J. A. Thomas, Elements of information theory. John Wiley & Sons, 2012.

[14] T. L. Griffiths and M. Steyvers, “Finding scientific topics,” Proceedings of the National academy of Sciences of the United States of America, vol. 101, no. Suppl 1, 2004, pp. 5228–5235.

[15] S. Haiduc and A. Marcus, “On the use of domain terms in source code,” in The 16th IEEE International Conference on Program Comprehension, ICPC 2008. IEEE, 2008, pp. 113–122.

[16] K. Somasundaram and G. C. Murphy, “Automatic categorization of bug reports using latent dirichlet allocation,” in Proceedings of the 5th India Software Engineering Conference. ACM, 2012, pp. 125–130.

[17] M. Alenezi, K. Magel, and S. Banitaan, “Efficient bug triaging using text mining,” Journal of Software, vol. 8, no. 9, 2013, pp. 2185–2190.

[18] M. Alenezi and K. Magel, “Empirical evaluation of a new coupling metric: Combining structural and semantic coupling,” International Journal of Computers and Applications, vol. 36, no. 1, 2014.

[19] X. Sun, B. Li, Y. Li, and Y. Chen, “What information in software historical repositories do we need to support software maintenance tasks? an approach based on topic model,” in Computer and Information Science. Springer, 2015, pp. 27–37.

[20] K. Herzig and A. Zeller, “The impact of tangled code changes,” in 10th IEEE Working Conference on Mining Software Repositories (MSR). IEEE, 2013, pp. 121–130.

[21] I. Keivanloo, C. Forbes, A. Hmood, M. Erfani, C. Neal, G. Peristerakis, and J. Rilling, “A linked data platform for mining software repositories,” in 9th IEEE Working Conference on Mining Software Repositories (MSR). IEEE, 2012, pp. 32–35.