Explore the Evolution of Development Topics via On-Line LDA

Jiajun Hu*, Xiaobing Sun*†, Bin Li*†,
*School of Information Engineering, Yangzhou University, Yangzhou, China
†State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
jiajunhu.yzu.edu@gmail.com, xbsun@yzu.edu.cn, lb@yzu.edu.cn

***Abstract*—Software repositories such as revision control systems and bug tracking systems are usually used to manage the changes of software projects. During software maintenance and evolution, software developers and stakeholders need to investigate these repositories to identify what tasks were worked on in a particular time interval and how much effort was devoted to them. A typical way of mining software repositories is to use topic analysis models, e.g., Latent Dirichlet Allocation (LDA), to identify and organize the underlying structure in software documents to understand the evolution of development topics. These previously LDA-based topic analysis models can capture either changes on the strength (popularity) of various development topics over time (i.e., strength evolution) or changes in the content (the words that form the topic) of existing topics over time (i.e., content evolution). Unfortunately, few techniques can capture both strength and content evolution simultaneously. However, both pieces of information are necessary for developers to fully understand how software evolves. In this paper, we propose a novel approach to analyze commit messages within a project's lifetime to capture both strength and content evolution simultaneously via Online Latent Dirichlet Allocation (On-Line LDA). Moreover, the proposed approach also provides an efficient way to detect emerging topics in real development iteration when a new feature request arrives at a particular time, thus helping project stakeholds progress their projects smoothly.**

I. Introduction

As the development of a software project progresses, its complexity grows accordingly, making it difficult to understand and maintain. During software maintenance and evolution, software developers and stakeholders often seek ways to know the history and evolution of certain aspects of the system to help understand software development. For example, project managers can understand what tasks have recently been working on and how much effort has been devoted to each task by retrieving revision control systems [1], [2], while developers can understand the evolution of certain features of source code by mining source code repository [3], [4].

However, statistical results showed that about 80% of the data in software repositories is unstructured (in the form of natural language text) [5], [6]. This leads to some fundamental challenges in mining software repositories because unstructured data is often unlabeled, vague, and noisy [7]. One of recent advanced techniques is to use topic analysis models, such as Latent Dirichlet Allocation (LDA) [8], to automatically extract topics from textual repositories to explore and organize the underlying structure of software documents. Based on topic analysis, software documents are represented by a mixture of topics and each topic is represented by a set of tokens extracted from corresponding documents. These tokens co-occur frequently in the entire corpus and usually have a close sematic relationship. Ideally, these extracted topics correlate well with actual themes in software development such as features in source code and development activities in commit messages, thus providing a simple way for project stakeholders to quickly understand software artifacts.

In the past, several LDA-based approaches have been proposed to aid developers in understanding software evolution. Thomas et al. applied *the Hall Model* [9] to analyze the history of source code documents to discover and monitor the drift of topics in source code [4], [10]. They applied LDA to all of the versions of the system at once, then mapped the topics back to individual version of the system. They computed several topic metrics (e.g., assignment or weight [11]) to represent the strength (popularity) of a topic for each version. In this way, the evolution of the strength of a topic (ideally correlates with real features in source code) can be monitored. However, the content (words that form a topic) of a topic never changes across the versions, which rarely occurs in practical software evolution. On the other hand, in order to monitor the day-to-day development activities, Hindle et al. applied *the Link Model* [12] to mine the history of commit messages to recover the development topics during a time interval [1]. They applied LDA to each time window of the corpus separately, then linked topics in different time windows together according to a similarity measure (for example, topics that share 8 out of 10 words across successive time windows). In this way, the evolving stream of development topics over time can be monitored by the change of the content of topics over time. However, the strength of a topic across all time windows could not be recovered which is also not fit for the practical fact during software evolution.

As discussed above, none of existing techniques can capture both strength and content evolution simultaneously. However, both pieces of information are necessary for developers to fully understand how software evolves. For example, project managers may hope to know how much work was related to feature *A* at a certain time period, which can be obtained by computing the corresponding topic strength at that time. What's more, they might also have the request to know what kind of work was done for feature *A* at a particular time point. Only the topic strength will not help project managers work

on this task. Instead, the content of the corresponding topic can shed light on activities that are performed for feature *A* at that time point.

To fill this gap, we propose a novel approach to capture both topic strength and content evolution of development topics simultaneously. We apply Online Latent Dirichlet Allocation (On-Line LDA) [13], which is an online version of LDA that incrementally builds an up-to-date model (mixture of topics per document and mixture of words per topic) when a set of new documents appears in the commit messages of a software repository. In order to monitor how much effort has been devoted to each development topic at a ceratin time, we measure the strength (popularity) of the corresponding topic by computing *Normalized-Assignment* metric at each time window and map the values to timestamps to represent *topic strength evolution*. Meanwhile, in order to better understand the detailed information of each development activity and its evolution, we track the content (words that form the topic) of the corresponding topics over time to capture *topic content evolution*. In addition, during real development iteration, new features and requests frequently appear to become emerging topics, old features and requests will disappear and become outliers. Automatically detecting these emerging topics can provide a meaningful view for project managers to monitor the progress of their project. By computing topic similarity measure (e.g., KL divergence [14]) based on the evolution matrix generated by On-Line LDA, these emerging topics can be easily detected. Compared with previously LDA-based approaches, the main advantages of our approach are listed as follows:

- We capture software evolution from the perspective of both topic strength and content evolution, thus providing a more complete and comprehensive view of software evolution.
- Rather than detecting what topics are relevant across time windows [1], we detect emerging topics at the time of their arrival, which is novel and different from the previous work.
- By applying On-Line LDA, the current model is incrementally updated according to the information inferred from the new set of data with no need to access the previously processed documents, which improves its memory usage and time complexity.

II. Background and Notation

In this section, we introduce the related topic analysis models and revision control system. A glossary of notations used in this paper is shown in Table I.

A. LDA

Latent Dirichlet Allocation (LDA) is a probabilistic technique that has been successfully used to extract the latent topics to describe a corpus of text documents [8] . The main idea behind LDA is that it models each document as a multi-membership mixture of K corpus-wide topics, and each topic as a multi-membership mixture of the terms in the corpus vocabulary. This means that there is a set of topics to describe the entire corpus and each document is composed of one or more topics. Each term in the vocabulary can be contained in more than one of these topics. Hence, LDA is able to discover a set of ideas or themes that well describe the entire corpus.

TABLE I
Notation used in the paper

SYMBOL	DESCRIPTION
K	number of topics
N	total number of unique words
N_d	number of words in document d
δ	size of sliding window
S^t	a stream of documents arriving at time t
M^t	number of documents arriving at time t
w_{di}^t	the unique word associated with the i^{th} token in document d at time t
z_k^t	the topic with index k at time t
θ_d^t	the multinomial distribution of topics specific to the document d at time t
ϕ_k^t	the multinomial distribution of words specific to the topic k at time t
α_d^t	K-vector of priors for document d at time t
β_k^t	N-vector of priors for topic k at time t
B_k^t	$N \times \delta$ evolution matrix of topic k with columns $= \phi_k^i, i \in \{t-\delta, ..., t\}$
ω^δ	δ-vector of weights of $\phi^i \in \{t-\delta, ..., t\}$

LDA is a fully generative model that describes how documents are created. Informally stated, this generative model makes the assumption that the corpus contains a set of K corpus-wide topics, and each document is composed of various combinations of these topics. Each term in each document comes from one of the topics in the document. This generative model is formulated as follows:

- Choose a topic vector $\theta_d \sim Dirichlet(\alpha_d)$
- Choose a word vector $\phi_k \sim Dirichlet(\beta_k)$
- For each word w_i in each document d:
  - Draw a topic $z_k \sim Multinomial(\theta_d); (p(z_i|\alpha))$
  - Draw a word $w_i \sim Multinomial(\phi_k); (p(w_i|z_k, \beta))$

Here, α and β are the priors for document-topic distributions and topic-word distributions respectively. A lot of estimation strategies for α and β, such as Gibbs sampling [15], are proposed in the machine learning literature.

B. On-Line LDA

On-Line LDA [13] is a topic model that automatically captures the underlying themes of text streams and their changes over time. It is an online version of LDA that incrementally builds an up-to-date model (mixture of topics per document and mixture of words per topic) when a set of new documents appears. On-Line LDA considers the temporal ordering information and assumes that the documents are divided by time slices. At each time slice, a topic model with K components is used to model the incoming documents. The generated model, at a give time, is used as prior for LDA at a successive time slice, when a new data stream is available for processing. Therefore, On-Line LDA allows to incorporate inferred semantics from the past data to guide the

inference process of the upcoming streams. This is achieved by considering all the topic-word distributions learned within a sliding "*history window*" when constructing the current priors.

To formulate the problem, let B_k^t denote an *evolutionary matrix* of topic k in which the columns are the topic-word distribution ϕ_k^j, which is generated for streams received within the time specified by the sliding history window δ, i.e., $j \in \{t-\delta, ..., t\}$. Let ω be a vector of δ weights, each of which is associated with a time slice from the past to determine its contribution in computing the priors for stream S^{t+1}. We assume that the sum of history weights $\sum_{t=1}^{\delta} \omega^t$ is equal to one. Hence, the parameters of a topic at time $t+1$ are determined by a weighted mixture of the topic's past distributions as follows:

$$\beta_k^{t+1} = B_k^t \omega \tag{1}$$

Under this definition of β, topic distributions in consecutive models are aligned so that the evolution of topics in a sequential corpus is captured. Thus, the generative model for time slice t is formulated as follows:

- For each document d
  - Draw $\theta_d^t \sim Dirichlet(\alpha^t)$
- For each topic $k = 1, ..., K$
  - Compute $\beta_k^t = B_k^{t-1}\omega$
  - Draw $\phi_k^t \sim Dirichlet(\beta_k^t)$
- For each word w_i in document d
  - Draw $z_k \sim Multinomial(\theta_d^t); (p(z_k|\alpha^t))$
  - Draw $w_i \sim Multinomial(\phi_k); (p(w_i|z_k, \beta_k^t))$

When *time slice*=1, the topic parameter, ϕ_k^1, is drawn from a Dirichlet prior, $Dir(\beta_k^1)$, where β_k^1 is initialized to be a constant, b, as done in the original LDA modeling [15].

C. Revision Control System

In this paper, the unstructured software repository to be analyzed is revision control system. A revision control system maintains and records the history of changes. Typically, developers use revision control systems to maintain the activities performed in their project. Most revision control systems (including CVS [16], SVN [17], and Git [18]) allow developers to enter a commit message when they commit a change into the repository, describing the change at a high level, i.e., who changed what and when. These unstructured commit messages are of grate value because they record the history of changes during the iterative development of a software project, thus describing how the project is evolving over time.

In this paper, our focus is to aid developers in monitoring the day-to-day development activities (e.g., what task has currently been working on and how it changes). Since the history of commit messages records the development activities committed by developers, meaningful and practical development topics can be extracted from such a repository. Meanwhile, when dividing the history of commit messages by time slices (e.g., a month, a season, or a year), it is much similar like a text stream for On-Line LDA to analyze the evolving themes.

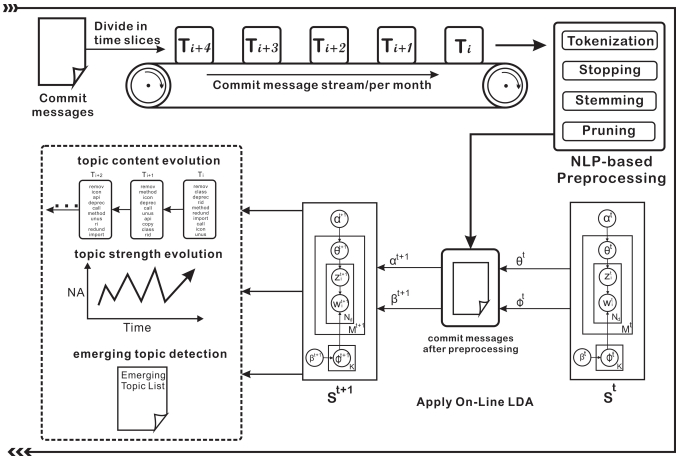

Fig. 1. The general steps for our approach

III. Approach

Figure 1 depicts the general process of our approach. First, commit messages are divided by time slices. Second, several data preprocessing operations are applied when a set of new messages arrives. Once the data is preprocessed, we apply On-Line LDA to construct the model. Finally, we measure the topic strength and content evolution, and detect emerging topics based on the On-Line LDA model.

A. Commit Messages Divided by Time Slices

The commit messages, which are ordered by their commit time, are included in a corpus with timestamps. We cannot apply On-Line LDA at the arrival of just one commit message because the granularity is too fine. Also, we cannot apply On-Line LDA until one year's accumulative commit messages come, in this situation, the granularity is too coarse. So choosing the optimal time slice is an essential task for On-Line LDA to infer meaningful topics. In our approach, we choose one month as the default time slice because it is smaller than the time between minor releases but large enough to have many commits to analyze [1].

B. NLP-based Preprocessing

We cannot apply On-Line LDA when a set of new documents appears without any preprocessing operations because there does exist some noise in the unstructured commit messages, which will confuse and distract the topic analysis tools. Natural language preprocessing (NLP) techniques are usually used to perform one or more preprocessing operations before applying topic models to reduce noise and improve the quality of the resulting text [19].

We preprocess the commit messages by applying typical natural language preprocessing operations. We first split the original commit messages into tokens and remove unrelated and unimportant words, such as the punctuation, numeric characters (e.g., @, *, !). We also remove common English language stop words (e.g., *the*, *it*, *in*) to reduce noise. Then, we stem each word to its original format (e.g., "replaced"

becomes "replac") to reduce vocabulary size. Finally, we prune the vocabulary by removing overly rare words (those that occur in less than ε times, which is a threshold relying on the size and type of the corpus) because these words are of little use for topic analysis.

C. Topic Strength and Content Evolution

Once we have a set of newly arrived and preprocessed commit messages, we apply On-Line LDA to extract K topics. We note that On-Line LDA is incrementally updated according to the information inferred from the new stream of data with no need to access the previously processed documents, which improves its memory usage and time complexity.

In order to answer the questions such as *How much effort has been devoted to feature A during project development iteration*, we track the strength (popularity) of the corresponding topics as a function of time. We quantify and measure how the topic strength changes over time by computing a *Normalized-Assignment* (*NA*) metric at each time point. The *NA* of the topic is the average value of the topic memberships of all documents in that topic at a time, which indicates the total presence of the topics throughout the messages in that time. A higher *NA* means that a large portion of the messages (i.e., a large portion of the development activities) is relevant to that topic. We define *NA* of topic z_k at time t as

$$NA(z_k^t) = \frac{\sum_{d=1}^{M^t} \theta_d^t[k]}{M^t} \tag{2}$$

The *strength evoluton* (*SE*) of a topic z_k is a time-indexed vector of *NA* values for that topic

$$SE(z_k) = [NA(z_k^1), NA(z_k^2), \cdots, NA(z_k^t))] \tag{3}$$

Having only the evolution of the effort devoted to a certain feature is far from enough, the detailed information about that feature may also experience changes over time. For example, most development activities may be focused on fixing bug *A* at time *t*. However, due to some reasons, the focus moves to fixing bug *B* at time *t+1*. In this situation, automatically detecting the dynamic changing content of topic *fixing bug* can shed light on the detailed activities that are performed for this task. We can track the change of the detailed information of a development activity by exploring the *content evolution* of corresponding topic k. We define the content of a topic k by using the top 10 most frequent words sorted by the topic word distribution ϕ_k because in that distribution, word memberships with high probability have a strong relationship with that topic. Further, the content evolution of a topic k can be viewed as the changed top 10 most frequent words according to $\phi_k^j, j \in \{1, 2, ..., t\}$.

D. Emerging Topics Detection

In real project's iterative development process, new features and requests frequently appear as emerging topics, meanwhile, old features and requests frequently disappear and gradually become outliers. Automatically detecting these emerging topics can provide a meaningful view for project managers to monitor their project's progress. For example, there is an urgent request *A* that arrives at time *t* for developers to resolve. If the project progresses smoothly, the activities related to that request ought to be reflected in the commit messages (so do the extracted development topics) at that time. Otherwise, the project managers may wonder what tasks developers are currently working on. Automatically detecting these emerging topics can help managers ease such a task.

In our approach, we detect emerging topics by computing topic similarity measure (i.e., KL divergence [14]) based on the evolution matrix B_k^t generated by On-Line LDA incrementally. The idea is to compute dissimilarity between two topics, which are tied and aligned across successive time (i.e., two continuous columns in B_k^t). If the dissimilarity exceeds a pre-determined threshold, the topic at later time is considered as an emerging topic. The dissimilarity between two topic distributions, p and q, can be computed by using Kullback Leibler (KL) divergence

$$KL(p \parallel q) = \sum_i p(i) log \frac{p(i)}{q(i)} \tag{4}$$

KL divergence is not a real metric, since it is not symmetric. Thus, in our work, we compute the average value between $KL(p \parallel q)$ and $KL(q \parallel p)$ and denote it as KL distance (D_{KL})

$$D_{KL}(p \parallel q) = \frac{KL(p \parallel q) + KL(q \parallel p)}{2} \tag{5}$$

Thus, to determine whether the topic k arriving at time j (i.e., z_k^j) is an emerging topic, we compute the dissimilarity between $B_k^t(:, j)$ (i.e., ϕ_k^j) and $B_k^t(:, j-1)$ (i.e., ϕ_k^{j-1}), i.e., $D_{KL}(B_k^t(:, j) \parallel B_k^t(:, j-1))$. If the dissimilarity is bigger than a pre-defined threshold, topic z_k^j will be merged into the emerging topic list. With the help of this information, project managers can be notified when new tasks occur in project development process. If current emerging topics correlate with real new tasks, project is progressing smoothly. Otherwise, project managers need to check what is the current focus of the project or what hampers the project.

IV. Evaluation Plan

The purpose of our approach is to aid project managers and stakeholders in understanding and monitoring the evolution of development activities. We are planning a set of exploratory and descriptive case studies aiming at the efficiency and effectiveness of the proposed approach. The following research questions of our approach will be studied.

- RQ1 How well does the discovered topic evolution correspond to actual development activities reflected in commit messages?
- RQ2 How well do the detected emerging topics at each time slice correspond to novel development activities that are different from pervious ones?
- RQ3 Can our approach provide a more complete and comprehensive view of software evolution than previous approaches?

For *RQ1*, we wish to determine whether the discovered topic strength and content evolution correspond to real changes in commit messages. We will conduct some empirical case studies on several real software systems (e.g., *jEdit* and *PostgreSQL*). We will choose a random subset of time slices and explore the extracted topics at each time. Also, we will investigate into the commit messages related to each topic to see whether the content and strength of that topic correlate well with real development activities.

For *RQ2*, we wish to evaluate the capability of our approach of detecting novel topics at the time of their arrival. We will select random samples of emerging topics from random time slices and compare each topic with the aligned topic from previous time. We will compare them from two aspects: (a) exploring the content of two topics, and (b) investigating into original messages related to each topic. Because only in this way can we deeply understand the meaning of a topic, otherwise, our understanding may be one-sided. If two topics are obviously dissimilar with each other in both aspects, we consider the topic at later time as a truly emerging topic.

For *RQ3*, our study will involve some participants from school and/or industry. These participants will be divided into three groups. They are assigned with a subset of commits extracted from real systems. Their task is to comprehend the development activities and their evolution reflected in commit messages. One group is the experimental group that uses the tool based on our proposed techniques i.e., their task will be facilitated by providing both topic strength and content evolution. The second group is the control group that uses the tool only with the help of strength evolution [4], [10]. The final group is the other control group that uses the tool only with the help of content evolution [1]. Then, we compare the time they spend in finishing the task and the quality of their results.

V. Conclusion and Future Work

Traditional topic evolution models were designed to produce either the strength evolution or content evolution of the unstructured software repositories. In some cases, developers may have the request to know not only the evolution of the strength of one topic but also the evolution of detailed information within it. In this paper, we proposed a novel approach based on On-Line LDA to understand software evolution at both two views, i.e., strength evolution and content evolution, simultaneously. In addition, emerging topics of interest can also be detected by On-Line LDA. Our future work will focus on the implementation of our approach and several empirical case studies will be conducted to evaluate the effectiveness of our approach.

VI. Acknowledgments

This work is supported partially by Natural Science Foundation of China under Grant No. 61402396, No. 61472343, No. 61402395 and No. 61472344, partially by the Natural Science Foundation of the Jiangsu Higher Education Institutions of China under Grant No. 13KJB520027, partially by the Open Funds of State Key Laboratory for Novel Software Technology of Nanjing University under Grant No. KFKT2014B13, partially by the Jiangsu Training Programs of Innovation and Entrepreneurship for Undergraduates under Grant No. 201411117066X, and partially by the Cultivating Fund for Science and Technology Innovation of Yangzhou University under Grant No. 2013CXJ025.

References

[1] 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.

[2] S. McIntosh, K. Legere, and A. E. Hassan, "Orchestrating change: An artistic representation of software evolution," in *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering, CSMR-WCRE 2014, Antwerp, Belgium, February 3-6, 2014*, 2014, pp. 348–352.

[3] A. E. Hassan, A. Hindle, P. Runeson, M. Shepperd, P. T. Devanbu, and S. Kim, "Roundtable: What's next in software analytics," *IEEE Software*, vol. 30, no. 4, pp. 53–56, 2013.

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

[5] R. Blumberg and S. Atre, "The problem with unstructured data," *DM REVIEW*, vol. 13, pp. 42–49, 2003.

[6] S. Grimes, "Unstructured data and the 80 percent rule," *Carabridge Bridgepoints*, 2008.

[7] A. E. Hassan, "The road ahead for mining software repositories," in *Frontiers of Software Maintenance, 2008. FoSM 2008.* IEEE, 2008, pp. 48–57.

[8] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *the Journal of machine Learning research*, vol. 3, pp. 993–1022, 2003.

[9] 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.

[10] S. W. Thomas, B. Adams, A. E. Hassan, and D. Blostein, "Studying software evolution using topic models," *Sci. Comput. Program.*, vol. 80, pp. 457–479, 2014.

[11] P. F. Baldi, C. V. Lopes, E. J. Linstead, and S. K. Bajracharya, "A theory of aspects as latent topics," in *ACM Sigplan Notices*, vol. 43, no. 10. ACM, 2008, pp. 543–562.

[12] 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.

[13] L. AlSumait, D. Barbará, and C. Domeniconi, "On-line lda: Adaptive topic models for mining text streams with applications to topic detection and tracking," in *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*. IEEE, 2008, pp. 3–12.

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

[15] 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, pp. 5228–5235, 2004.

[16] B. Berliner *et al.*, "Cvs ii: Parallelizing software development," in *Proceedings of the USENIX Winter 1990 Technical Conference*, vol. 341, 1990, p. 352.

[17] C. M. Pilato, B. Collins-Sussman, and B. W. Fitzpatrick, *Version control with subversion*. " O'Reilly Media, Inc.", 2008.

[18] C. Bird, P. C. Rigby, E. T. Barr, D. J. Hamilton, D. M. German, and P. Devanbu, "The promises and perils of mining git," in *Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on*. IEEE, 2009, pp. 1–10.

[19] X. Sun, X. Liu, J. Hu, and J. Zhu, "Empirical studies on the nlp techniques for source code data preprocessing," in *Proceedings of the 2014 3rd International Workshop on Evidential Assessment of Software Technologies*, ser. EAST 2014, 2014, pp. 32–39.