DiffLDA: Topic Evolution in Software Projects [Technical Report 2010-574]

July 2010

Stephen W. Thomas, Bram Adams, Ahmed E. Hassan, and Dorothea Blostein

Software Analysis and Intelligence Lab (SAIL)
School of Computing
Queen's University
Kingston, Ontario, Canada

Abstract

Previous research has shown that *topics* can be automatically discovered in a software project's source code. Topics are collections of words that co-occur frequently in a text collection and are discovered using topic models such as latent Dirichlet allocation (LDA). Tracking how topics evolve, i.e., grow and spread, over time is useful for supporting software maintenance, comprehension, and re-engineering activities.

The evolution of topics is typically recovered by applying LDA to all versions of a project's source code at once, followed by post processing to map topics across versions. Although this technique works well in applications where each version of the data is completely different, for example in the analysis of conference proceedings, the technique does not work well with source code, which typically changes only incrementally and contains significant duplication across versions. In this paper, we present a new approach, called DiffLDA, for automatically mining topic evolution in source code. The approach addresses LDA's sensitivity to document duplication by operating on the differences between versions of a source code document, resulting in a more accurate, finer-grained representation of topic evolution. We validate our approach through case studies on simulated data and two open source projects.

1 Introduction

Topics are collections of words that co-occur frequently in a corpus of text documents, and can be used to provide structure to an otherwise unstructured collection of text [1]. *Topic models*, such as latent Dirichet allocation (LDA) [2], are statistical models used to automatically extract the topics from a given corpus and have proven to be an effective tool for analyzing, understanding, and describing software project artifacts [3–6].

Recently, understanding how topics evolve over time in time-stamped or versioned text collections has gained the attention of researchers, as it promises to be an effective way to study the history and evolution of the collection [7–10].

However, traditional topic evolution models have the unstated assumption that the documents are unique at each time period, i.e., there is no duplication in the collection. However, this assumption does not hold in software projects, where documents (e.g., source code files or requirements documents) are updated incrementally and contain significant duplication across time periods. For example, many source code documents do not change at all between versions of a software system, and the documents that do change are usually only incrementally fixed or updated, not written from scratch. Requirements documents may receive small corrections and updates between versions, but remain largely the same over time. Even mailing list messages often contain quoted copies of previous messages to provide context for the recipients. In this paper, we show that all of this duplication poses serious problems for traditional topic evolution models and greatly affects the quality of their results.

To combat the duplication issue, we propose *DiffLDA*, a novel topic evolution model that explicitly captures the edits between versions, effectively eliminating the duplication issue limiting traditional LDA. DiffLDA preprocesses the collection of versioned documents to compute the changes between each version, applies LDA only to the changes, and post processes the results to reconstruct the original documents. In this paper we show that such an approach results in improved precision and recall of topic evolutions as well as reduced execution time over traditional approaches.

The contributions of this paper can be summarized as follows:

1. We show that existing topic evolution models based on LDA do not handle data duplication well. Such duplication leads misleading topic evolution metrics.

2. We present the DiffLDA approach, which can overcome the data duplication issue.

3. We present case studies on simulated and real-world projects using DiffLDA to demonstrate its effectiveness.

The rest of this paper is organized as follows. Section 2 provides a brief background of LDA and how topic modeling has been used in software maintenance activities. Section 3 motivates our approach with an example. Section 4 formalizes our proposed approach, and Section 5 validates our approach on simulated and real world data. Section 6 weighs the pros and cons of our approach and Section 7 concludes.

2 Background and Related Work

In this section we describe latent Dirichlet allocation, briefly outline how topic models have thus far been applied to software projects for maintenance and comprehension tasks, and describe the state-of-the-art of topic evolutions models.

2.1 Latent Dirichlet Allocation

LDA is a popular probabilistic topic modeling technique [2]. *Topic modeling* is an automated technique designed to discover *topics* within a corpus of text documents [1], where topics are collections of words that co-occur frequently in the corpus. Due to the nature of language usage, the words that constitute a topic are often semantically related. Documents can be represented by the topics within them, and the entire otherwise unstructured corpus can be structured in terms of this discovered semantic structure.

LDA models each document in a corpus as a multi-membership mixture of T topics, and each topic as a multi-membership mixture of the words in the corpus vocabulary. A multi-membership mixture means that each document can contain more than one topic, and each word can be contained in more than one topic. Hence, LDA is able to discover a set of ideas or themes that well describe the corpus as a whole [1].

More formally, LDA produces, for each of T topics, an N-dimensional word membership vector $z(\phi_{1:N})$ that describes which words appear in topic z, and to what extent. Additionally, for each document d in the corpus, LDA produces a T-dimensional topic membership vector $d(\theta_{1:T})$ that describes the extent to which each topic appears in d.

2.1.1 Metrics

It is possible to compute a ***topic similarity*** between two topics z_1 and z_2 by computing the KL divergence [11] between their word membership vectors [12]:

$$\mathrm{KL}(z_1(\phi), z_2(\phi)) = \sum_{i=1}^{N} z_1(\phi_i) \log \frac{z_1(\phi_i)}{z_2(\phi_i)}.$$

It is also possible to compute a ***document similarity*** of two documents d_1 and d_2 by computing the Hellinger distance between their topic membership vectors [1]:

$$\mathrm{HD}(d_1(\theta), d_2(\theta)) = \sum_{t=1}^{T} (\sqrt{d_1(\theta_t)} - \sqrt{d_2(\theta_t)})^2.$$

We can compute the ***assignment*** of a topic z_k across a system by summing the membership values of that topic over all documents,

$$\mathrm{A}(z_k) = \sum_i d_i(\theta_k),$$

which gives a good indication of the total volume of the topic throughout the corpus.

2.1.2 Considerations

We note that in LDA, the number of topics T to be discovered is an input into the model, along with document and topic smoothing parameters α and β. As there is no standard method for choosing the values for these input parameters beforehand, in this paper we use established heuristics that have been shown to have reasonable performance [13,14]. An alternative approach is to first learn the number of topics using algorithms such as the Hierarchical Dirichlet Process [15].

We also note that LDA is a generative probabilistic model in which exact inference is intractable, and Gibbs sampling is often used to sample the posterior probabilities of documents and topics. As such, different sets of sampling iterations will produce slightly different results, i.e., applying LDA to a data set is a non-deterministic process.

2.2 Topic Modeling in Software Maintenance

LDA and related topic modeling techniques have seen widespread adoption in software engineering, in particular in the software maintenance field.

Maskeri et al. used LDA on a single snapshot of the source code of a software project to extract the business topics and their locations in the system, with the goal of easing the comprehension of large systems for newcomers [3].

Latent Semantic Indexing (LSI) [5], another topic modeling technique, has been used to incrementally update the traceability links between software artifacts as a software system evolves over time by utilizing the results from previous versions during the computation of the current version [16].

Poshyvanyk et al. combined LSI and formal concept analysis (FCA) to build a *concept lattice* that enables concept location [17]. A user or developer can then throw queries to the concept lattice to locate relevant source code topics of interest.

Kuhn et al. introduced *semantic clustering*, a technique based on LSI to group source code documents that share a similar vocabulary [6]. After applying LSI to the source code, the documents are clustered based on their underlying topical structure into semantic clusters, resulting in clusters of documents that implement similar functionalities.

LDA has also been used to uncover *software concerns*, which are usually defined as anything that a stakeholder considers as a conceptual unit [18], such as high-level business logic, IO handling, GUI interaction, and authentication [4].

Hindle et al. applied a windowed topic analysis approach, based on LDA, to log messages from a project's version control system [19]. Hindle et al. also presented several visualization techniques to understand the results. They argued that a windowed topic analysis provides the ability to highlight local topics as well as to identify global trends, and they associate local topics over time by using a top-word similarity measure.

2.3 Topic Evolution Models

Techniques have been established to automatically mine topics from a single snapshot of a software project using, for example, basic LDA [3, 4]. However, mining the evolution of topics across snapshots is still a challenge. Several techniques have been proposed in the natural language processing community for modeling the evolution of topics on other types of data:

The Dynamic Topic Model [7] models time as a discrete Markov process, where topics evolve according to a normal distribution. Such a model penalizes abrupt changes between successive time periods, making the model inappropriate for source code data, where the changes to individual topics between time periods could be dramatic—much more than a normal distribution would allow (see below for details).

The Topics Over Time (TOT) [8] approach models time as a continuous beta distribution, effectively removing the penalty on abrupt changes from the Dynamic Topic Model. However, the beta distribution is still too inflexible for source code, since it assumes that a topic evolution will have only a single rise and fall during the entire project history.

Hall et al. [9] apply LDA to the entire collection of documents at once and perform post hoc calculations based on the observed probability of each document in order to map topics to versions. Linstead et al. also used this approach on a software system's version history [10]. The main advantage of this approach is that no constraints are placed on the evolution of topics, yielding excellent flexibility for describing large, seemingly random changes to a corpus, which are typical in software development. We note that the Hall approach works very well for the type of data it was designed for (i.e., conference proceedings).

In this paper, we only consider the Hall et al. approach, since implementations are currently unavailable for the Dynamic Topic Model and Topics Over Time approaches (even after contacting the authors directly). We do not expect our conclusions to be affected, as we expect the Hall et al. approach to perform best on source code histories due the the lack of restrictions in the approach compared to the other approaches.

	Linux	**Eclipse**	**PSQL**
Number of versions	105	22	50
SLOC in latest version	4.8M	2.9M	501K
Number of files in latest version	8K	19K	884
Median % of files changed per version	0.1	2.9	7.0
Median % of lines changed per file	0.1	1.0	1.0
Median lifetime of each file (versions)	104	10	33

Table 1: Change characteristics of three software projects.

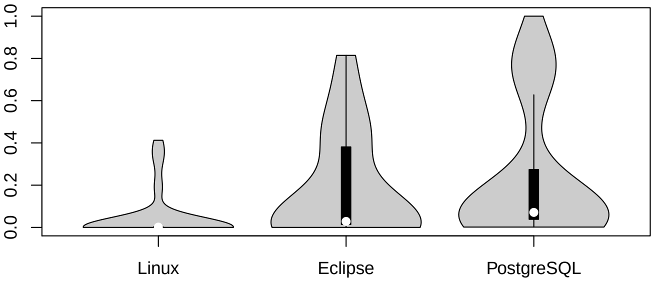


Figure 1: A violin plot of the percentage of source code documents changed between releases, for three different software projects. The white circles show the median values, the black bars show the interquartile range, and the gray shapes give an indication of the distribution.

3 Problem Statement

Understanding how a topic evolves over time greatly improves several software development tasks, such as maintenance, re-engineering, and program comprehension. For example, because of refactoring and other development efforts that occur over time, a feature or concern is likely to become more scattered across the source code documents, which affects the developers' ability to locate, understand, and modify the source code related to the feature or concern of interest. By monitoring the scattering of topics over time, developers and managers can make maintenance decisions, such as how to propagate changes to all the right locations or determining when a refactoring is necessary. Topic evolution analysis can also benefit the understanding and organization of mailing list discussions, for example by showing project stakeholders which topics were active during a given time period. Additionally, by studying how topics evolve, we can provide new insights into the history of software projects and the quality of certain initial designs.

Existing topic evolution models have the unstated assumption that the documents in the corpus are not duplicated, i.e., each document is unique.

This assumption holds for journals, blog posts, and newspaper articles, which are typically studied in the topic modeling literature. However, software modules are usually updated incrementally and see tremendous overlap between versions, although they occasionally see dramatic changes due to refactorings, and the addition or removal of functionalities.

To quantify the extent that source code breaks the assumption of current topic evolution models, we studied the change characteristics of three open source systems. Table 1 summarizes our findings. We found that on average, only 0.1%–7.0% of the source code documents experience any change between versions, measured by the number of files that had activity in the corresponding Version Control System (VCS). In other words, on average *at least 93% of the source code files are exact copies from version to version.* Furthermore, the changed files on average only had 0.1%–1.0% of their lines changed, as measured from the actual content of the VCS entry for each change. Finally, each source code file has a relatively long median lifetime, lasting for almost half or more of the lifetime of the software system.

However, the distributions in the violin plot [20] in Figure 1 show that even though the change behavior of source code is usually infrequent and small, there are periods with very large changes. (A violin plot shows a box-plot along with a kernel density plot.) For example, Figure 1 shows that Linux has on average only 0.1% of its source code files change between versions, but on one occasion over 40% of the files changed. Such scenarios can arise due to large refactorings, significantly new features that were added, or important maintenance activities.

To illustrate how LDA performs with the change characteristics of source code, we simulate analogous characteristics on the textual data studied by Hall et al [9], the ACL Anthology Reference Corpus [21]. The ACL corpus contains 10,920 conference papers between the years 1979–2004, consisting of a total of 556,372,830 tokens, 20,465 of which are unique. We apply the standard LDA preprocessing steps, such as removing common English-language stop words, removing words with one or two characters, and removing words that occur less than 10 times in the entire corpus. Congruent with Hall's approach, we model the corpus with $T = 46$ topics.

Scenario A, used as a baseline, contains the original data. Scenario B, used to illustrate the limitations of LDA, contains the original data along with several duplications of a single, randomly-chosen document. We selected `C96-2099`, which is an article from 1996 titled "Segmenting Sentences into Linky Strings" that describes a method to segment sentences based on patterns learned through a training corpus, with a case study on a Japanese-language corpus. We create 30 copies of this document, and apply LDA to the new corpus.

To compare the two scenarios, we will answer the following two questions:

1. What happens to the topic memberships of the document that is du-

Membership	ID	Top Words
Scenario A		
0.41	27	*japanese method case sentence sentences*
0.20	41	*word segmentation words character characters*
0.09	37	*word text list system dictionary string number*
Scenario B		
1.0	7	*similarity clustering cluster vector matrix*

Table 2: Top topics for `C96-2099` in the two scenarios.

plicated in Scenario B? Are its topic memberships the same as in Scenario A? We might initially expect that the topic memberships would be identical, since the content of the duplicated documents is identical.

2. Is the duplicated document related to the same set of documents in both scenarios? We use the Hellinger distance similarity for our study. Again, we might expect that since the content of the document is not changing between scenarios, the duplicated document should be related to the same set of documents.

3.0.1 Analysis of the Topic Memberships

Table 2 shows the most present topics of `C96-2099` after running LDA in both scenarios. In Scenario A, the majority of `C96-2099` is represented by three topics that deal with Japanese sentences, word segmentation, and dictionaries, respectively. In Scenario B, however, we see that `C96-2099` is now only described by a single topic, and we found that no other documents contained this topic with a membership greater than 0.05.

3.0.2 Analysis of the Related Documents

Table 3 shows the top three documents related to `C96-2099`, ranked by their Hellinger distance measure. We find that there is significant dissimilarity between Scenario A and Scenario B, as no related documents overlap. Further, the closest documents in Scenario B are more than twice as far from `C96-2099`, as compared to Scenario A.

Document duplication significantly impacts the results of LDA, and hence the state-of-the-art topic evolution techniques.

Document	HD	Title
Scenario A		
W02-1812	0.228	A Word Segmentation Method With Dynamic Adapting To Text Using Inductive Learning
C94-1036	0.284	Segmenting A Sentence Into Morphemes Using Statistic Information Between Words
C00-1084	0.288	Automatic Semantic Sequence Extraction from Unrestricted Non-Tagged Texts
Scenario B		
P87-1023	0.586	Now Let's Talk About Now: Identifying Cue Phrases Intonationally
W96-0308	0.586	Lexical Rules is Italicized
P81-1033	0.586	A Construction-specific Approach to Focused Interaction in Flexible Parsing

Table 3: The top three documents related to `C96-2099`.

4 Approaches

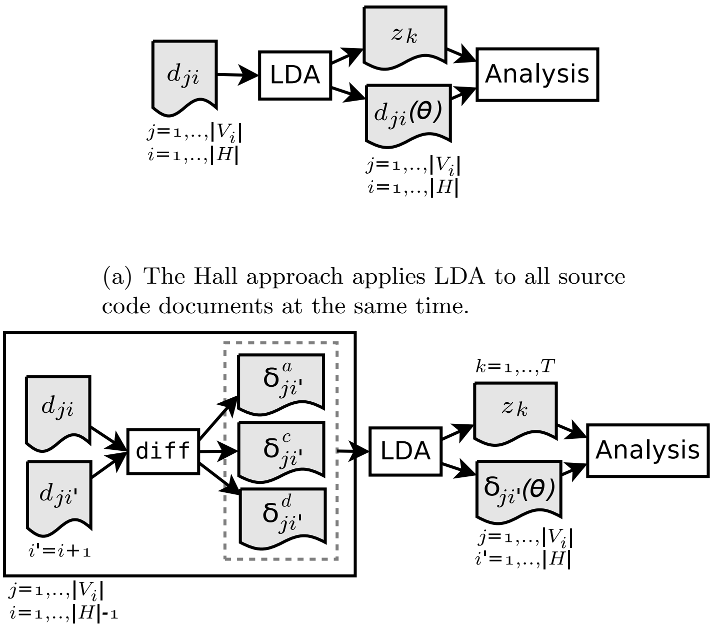
In this section we present a new approach for tracking the evolution of topics that deals with the duplication problem. Since our approach builds on the approach described by Hall et al. (henceforth the *Hall approach*), we first provide an overview of the Hall approach, followed by our proposed approach, which we name *DiffLDA*. We first provide a formalization of topic evolution modeling.

4.1 Formalization

A ***topic*** z is a tuple $\langle \mathbf{t}, \phi \rangle$ discovered by LDA, where $\mathbf{t}$ is the set of top terms related to the topic and ϕ is the normalized word distribution over the vocabulary.

A ***document*** d in the system is represented by $\langle n, sp, \mathbf{w}, \theta \rangle$, where n is the name of the document, sp is the subpackage to which the document belongs, $\mathbf{w}$ is the preprocessed content of the document (i.e., bag of words), and θ is the topic membership vector discovered by LDA.

A ***version*** V of a system is a set of documents $\{d_1, d_2, ...\}$ with the same time-stamp $t(V)$. The ***history*** H of a system is the set of versions related to the system, $H = \{V_1, V_2, ...\}$.



(a) The Hall approach applies LDA to all source code documents at the same time.

(b) DiffLDA adds a preprocessing step to the Hall approach to explicitly capture the edits.

Figure 2: A graphical depiction of the Hall and DiffLDA approaches.

Finally, the ***evolution*** E of a metric m of a topic z_k is a time-indexed vector of metric values for that topic: $E(z_k) = [m(z_k, t(V_1)), m(z_k, t(V_2)), ..., m(z_k, t(V_{|H|}))]$. Unless otherwise indicated, the metric m under consideration is the assignment metric.

In the remaining sections we use the following notation. d_{ji} is the j^{th} document in version V_i. $d_j(\theta_k)$ is the membership value of document d_j in topic z_k. We say that there are $|V_i|$ documents in a particular version V_i, and a total of $|H|$ versions in the system. To be consistent with the topic modeling literature, we use T to describe the number of topics in a system.

4.2 Hall Approach

Figure 2(a) provides an overview of the Hall approach. Here, LDA is applied to all source code documents in every version at once, followed by a post processing analysis phase to compute the metrics of interest. During the analysis phase, the assignment of a topic z_k at version V_i is computed as the aggregation of the topic's memberships across all documents in the specified version:

$$\mathrm{A}(z_k, V_i) = \sum_{j=1}^{|V_i|} d_{ji}(\theta_k).$$

For example, if 100 documents in version V_i each contained a membership of 0.5 for topic z_k, we would say that z_k has an assignment of 50 in V_i. Note that the Hall approach yields an assignment measure with a granularity at the document level.

4.3 DiffLDA Approach

In order to address the duplication issues in the Hall approach, DiffLDA prepends a preprocessing step that only propagates the changes between successive versions of each document, instead of the whole document at each version. Figure 2(b) depicts our approach.

For each source code document d_j in the system, we first compute the edits between successive versions V_i and $V_{i'}$, $i' = i + 1$ using the standard `diff` utility. We classify each edit as *add*, *change*, or *delete*, depending on whether the edit resulted in more, the same, or fewer lines of code, respectively. For each version of each document, we create three *delta documents*, $\delta^a_{ji'}$, $\delta^c_{ji'}$, and $\delta^d_{ji'}$, to capture these three types of edits. We place all the lines added between d_{ji} and $d_{ji'}$ into $\delta^a_{ji'}$, all the changed lines in $\delta^c_{ji'}$, and all the deleted lines in $\delta^d_{ji'}$. We use the notation $|\delta^a_{ji'}|$ to represent the number of words in the $\delta^a_{ji'}$ delta document. The notation $\delta^x_{ji}(\theta_k)$ represents the membership of topic z_k in delta document δ^x_{ji} for $x \in \{a, c, d\}$. We must handle two special cases: 1) When we first encounter a document d_j at version

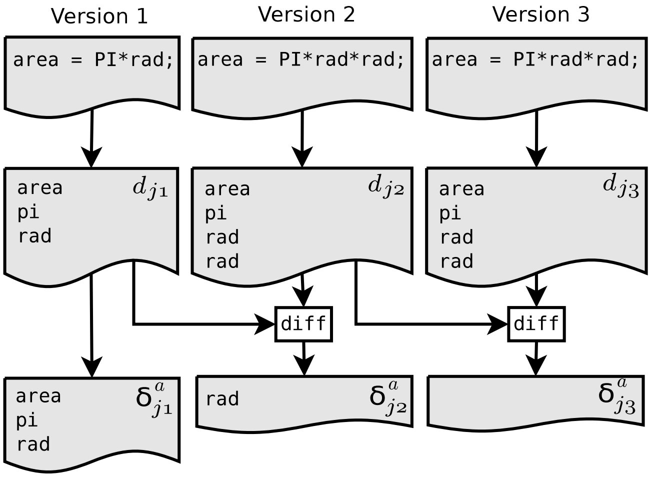


Figure 3: An example of three versions of a document. The Hall approach applies LDA to $\{d_{j1}, d_{j2}, d_{j3}\}$, which are the results of standard preprocessing. DiffLDA applies LDA to $\{\delta^a_{j1}, \delta^a_{j2}, \delta^a_{j3}\}$, which removes duplication.

V_i (either because $i = 1$ or d_j is a new document), we consider the entire document as additional words, and thus we add the entire document to the delta document δ^a_{ji}; and 2) when a document d_j is removed at version V_i, we consider the entire document as deleted words, and thus add the entire document to the delta document δ^d_{ji}.

Next, we apply LDA to the entire set of delta documents at once, resulting in a set of discovered topics and membership values for each delta document.

Finally, we examine the output of LDA to compute the metrics of interest. The corresponding assignment metric of a topic z_k at version V_i for the DiffLDA method is defined recursively as

$$\mathrm{A}(z_k, V_i) = \mathrm{A}(z_k, V_{i-1}) + \sum_{j=1}^{|V_i|} (\delta^a_{ji}(\theta_k) \left|\delta^a_{ji}\right| - \delta^d_{ji}(\theta_k) \left|\delta^d_{ji}\right|).$$

For each subsequent version in a system, the assignment metric is increased by the topic memberships of the words that were added in the new version, and decreased by the topic memberships of the words that were deleted. This cumulative addition is necessary since we only model the edits at each version.

We emphasize that the multiplication of the topic membership, $\delta_{ji}(\theta_k)$, by the number of words in the delta document, $\delta_{ji}(n)$, gives a very fine-grained view of the topic evolution: we are computing the assignment of each topic at the word granularity, whereas the Hall method computes the assignment of each topic at the document granularity.

Even though the change delta documents are not used in the computation of the assignment metric, the change delta documents are useful during the computation of other metrics, such as a ***hotness*** metric, which represents how much edit activity a topic z_k has received in version V_i:

$$\mathrm{H}(z_k, V_i) = \sum_{j=1}^{|V_i|} (\delta^a_{ji}(\theta_k) \left|\delta^a_{ji}\right| + \delta^d_{ji}(\theta_k) \left|\delta^d_{ji}\right| + \delta^c_{ji}(\theta_k) \left|\delta^c_{ji}\right|).$$

Figure 3 shows an example scenario with one document changing over three versions. The Hall approach will use d_{j1}, d_{j2}, and d_{j3} (middle row) for the LDA execution, while DiffLDA uses $\delta^a_{j1}, \delta^a_{j2}$, and δ^a_{j3} (bottom row). The δ^a_{j2} document only contains the single word that changed from version 1 to version 2 (`rad`), and δ^a_{j3} is empty since there are no changes between version 2 and version 3.

4.4 Discussion

In the Hall approach, misleading evolutions can occur due to the coarser granularity of the approach. For example, suppose that the assignment of a topic z_k at version V_i has a value of 10.0. If a new document d_j, with $d_j(\theta_k) = 1.0$, were added into the system at version V_{i+1}, the value of the assignment metric would become 11.0, *regardless of the size of* d_j. That is, it would not matter whether d_j contained a single line of code or one million lines of code—the assignment metric would increase by one unit. On the other hand, the finer granularity of the DiffLDA approach captures the size of additions and deletions to the topics in terms of individual words. In the example above, the assignment metric would increase by $1.0 \times |\delta^a_j|$, such that if $|\delta^a_j|$ is small, the assignment would not be affected much.

5 Empirical Validation of DiffLDA

Due to a lack of ground truth or benchmarks for the evaluation of topic evolution mining techniques, comparing the output of the Hall and DiffLDA approaches is difficult and somewhat subjective. Even matching a discovered topic evolution to existing documentation (e.g., change logs and release notes) does not guarantee an accurate evaluation, as documentation is often incomplete, vague, or out of date. Instead, we validate our approach by first applying it to a simulated (but well-understood) software project, followed by a more high-level analysis on two case studies on real-world (but less-understood) software projects.

5.1 Study Setup

For each simulated scenario and case study, we preprocess the data as usual for topic modeling by isolating identifiers and comments, tokenizing words based on common naming schemes such as `firstSecond` and `first_second`, and removing common English-language stopwords such as “the”, “it” and “on”. We use the MALLET tool [22] for the LDA computation, running for 100,000 Gibbs sampling iterations, the first half of which are used for parameter optimization [13]. We used standard techniques for determining the input parameters to LDA [13, 14]. The scenarios were executed on a machine running Ubuntu 9.10 with a 2.8GHz 16-core Intel Xeon CPU and 64Gb of main memory.

5.2 Evaluation Measures

We manually inspect the results of the Hall and DiffLDA approaches to evaluate the quality of their results. We examine each change event in the evolutions, where a *change event* can be an increase in a metric value (i.e., *spike*), decrease in a metric value (i.e., *drop*), or no change in a metric value (i.e., *stay*) between successive versions of a document.

We classify a change event as a spike or drop, respectively, if there is at least a 10% increase or decrease in metric value compared to the previous time period, and as a stay otherwise. Formally, for a metric m of topic z_k at version V_i, the change $c = \frac{m(z_k,V_i)-m(z_k,V_{i-1})}{m(z_k,V_{i-1})}$ is classified as

$$\text{Event}(z_k, V_i) = \begin{cases} \text{spike} & \text{if } c \geq 0.10 \\ \text{drop} & \text{if } c \leq -0.10 \\ \text{stay} & \text{otherwise.} \end{cases}$$

Thus, if an evolution contains 10 versions, then there are 10 change events. For each change event identified by an approach, we investigate existing documentation (for example, release notes, change logs, and source code), looking for any notes about the topic that caused the event. If we find any obvious mention of the topic in at least one documentation source, we interpret the event as correct, otherwise we say that it is incorrect.

We look for the following three desirable properties:

1. **Precision:** *Of all the change events to a discovered evolution, how many are correct?*

   We calculate the precision of a discovered evolution $E(z_k)$ as

   $$P(E(z_k)) = \frac{|\{\text{Correct events in } E(z_k)\}|}{|\{\text{Correct and incorrect events in } E(z_k)\}|}.$$

   In the special case that an evolution detected no events (for example, when no spikes were discovered), we set the precision to 1.0.

2. **Recall:** *Of all the actual (i.e., truth) events in the topic evolution, how many were discovered?*

We calculate the recall of a discovered evolution $E(z_k)$ as

$$R(E(z_k)) = \frac{|\{\text{Correct events in } E(z_k)\}|}{|\{\text{Truth events}\}|}.$$

In the case when there are no truth events (for example, when no spikes occur in a topic evolution), we set the recall the N/A.

3. **Speed:** *How much total execution time is required?*

   To compare the run-time speed of the Hall and DiffLDA approaches, we collect the execution times of the two stages of execution (i.e., preprocessing and LDA execution). We execute three repetitions of each scenario and case study and report the minimum times in effort to counter the effect of background processes running on the machine.

5.3 In-Depth Study of Simulated Data

In order to compare in detail the results of DiffLDA to those of the Hall approach, we create a simulated software project based on the `backend.access` subpackage of PostgreSQL. The `backend.access` subpackage contains 58 documents and 8 subdirectories, and is responsible for implementing various functionalities, including hash tables, transactions, and NBTrees.

Our goal is to manually create scenarios with a representative variety of specific events to the source code at specific versions, so we can determine whether the evolutions discovered by our approach are valid.

Scenario Descriptions: We create three scenarios of increasing complexity.

Scenario 1: This scenario contains 10 copies of the `backend.access` subpackage, with modified timestamps from 1–10. That is, all documents in version 1 are the same as their counterparts in version 2, which are the same as their counterparts in version 3, etc.

Since no changes are taking place to the source code documents, we expect this scenario to show no evolution of any of the discovered topics: we expect the assignment metrics to remain constant over the 10 versions.

Scenario 2: This scenario is the same as Scenario 1, with the exception that three documents from the unrelated `timezone` subpackage of PostgreSQL are artificially inserted at version 5, and removed at version 6.

We expect to see a sharp increase in assignment in the time zone related topic(s) at version 5, a sharp decline at version 6, and no change in assignment to any other topic.

Scenario 3: This scenario is the same as Scenario 2 with the following two additions: 1) Eight documents from the `ecpglib` subpackage of PostgreSQL are inserted in versions 9 and 10. The first half of each document is inserted in version 9, while the second half of each document is inserted at version 10. 2) Five documents from the `backend.regex` subpackage of PostgreSQL are inserted in version 1, they remain in versions 2 and 3, and are removed at version 4.

As in Scenario 2, we expect to see a sharp increase in the assignment metric for the time zone related topic(s) at version 5, followed by a sharp decline at version 6. Additionally, we expect to see a steady increase in the `ecpglib` related topic(s) over version 9 and 10, as well as a sharp decrease in the `backend.regex` related topic(s) at version 4.

We have made both scenarios publicly available on the first author's web page.

Results: Figure 4 shows the evolutions identified by both approaches. For Scenario 1, the figure shows that the Hall approach results in topics with somewhat noisy evolutions: even though the source code does not change between versions, the evolutions produced indicate that some changes are occurring. On the other hand, DiffLDA correctly shows that no evolution is occurring to any of the topics, shown by the flat evolution lines of each topic.

For Scenario 2, Figure 4 shows that the Hall approach again produces noisy topic evolutions. In this scenario, three unrelated files are inserted at version 5 and no other changes are made to the source code. While the Hall approach was able to capture the time zone-related spike at version 5 and drop at version 6, the same evolution also experiences spikes at versions 3 and 10 and a drop at version 4, when no changes actually occurred to the source code in those versions. The set of top words for this topic is "*state printtup slot buf attr typeinfo natts tuple info portal*", which is not an intuitive set of words for a time zone related topic. Indeed, we found that this topic is contained in 20 documents, instead of just the three inserted `timezone` documents. On the other hand, Figure 4 indicates that DiffLDA correctly shows topic activity for only one topic. This activity is represented by a large spike at version 5, followed by a drop at version 6. The set of top words for this topic is "*time strp year ptlim ttisp day isdst gmtoff offset week zone*" which is a more revealing list for the `timezone` documents. Further, we found that this topic is only contained in the three `timezone` documents.

For Scenario 3, Figure 4 shows again that the Hall approach produces noisy topic evolutions. This time, the Hall approach is able to correctly identify a topic evolution with a spike at version 5, a drop at version 6, and no other activity. However, the Hall approach confuses the other two changes (i.e., the `backend.regex` documents in versions 1, 2, and 3, and the `ecpglib` documents at versions 9 and 10) and assigns them to the same topic. Additionally, the expected gradual change at versions 9 and 10 are not shown; we instead see an abrupt change at version 9, followed by an slight decrease at version 10. On the other hand, DiffLDA correctly captures all three changes: the time-zone topic spikes and drops at versions 5 and 6, respectively; two topics related to the `backend.regex` documents drop at version 4, and the topic related to the `ecpglib` documents gradually

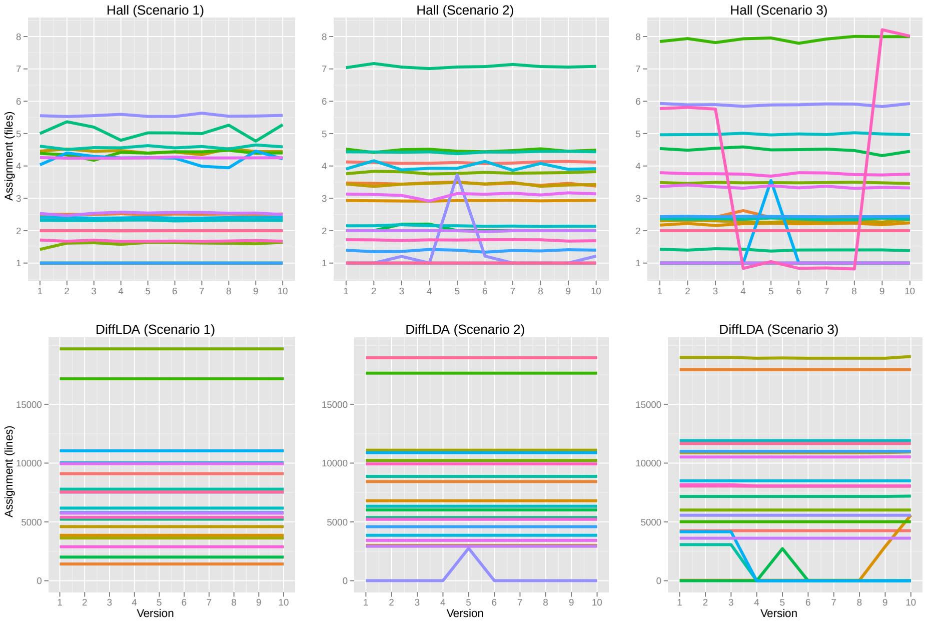


Figure 4: The topic evolutions of the three simulated scenarios, as produced by the Hall and DiffLDA approaches.

increases at versions 9 and 10.

Table 4 summarizes the results for precision, recall, and speed for each approach and each scenario. In Scenario 1, the Hall approach discovered several spikes and drops (namely at versions 2, 4, and 9) when none actually occurred, and thus receives a precision of 0.0 for these changes. DiffLDA discovered all 200 events correctly. We also see that DiffLDA has an execution time that is an order of magnitude faster than Hall's. This is because DiffLDA results in far less data, resulting in less processing time in the LDA execution phase.

The results for Scenario 2 are similar: the Hall approach discovered several spike and drop events that were incorrect, yielding low precision scores, while DiffLDA correctly discovered all events; DiffLDA also ran an order of magnitude faster than the Hall approach.

Scenario 3 saw similar results: the Hall approach discovered several incorrect events and ran an order of magnitude slower than DiffLDA. DiffLDA discovered only one incorrect spike event (a slight peak in one of the constant topics at version 10), resulting in a slightly penalized spike precision and stay recall.

On our simulated data, DiffLDA improves the precision and recall metrics while decreasing run time.

	Hall	**DiffLDA**
Scenario 1		
Spikes	$P = 0.0, R =$ N/A	$P = 1.0, R =$ N/A
Drops	$P = 0.0, R =$ N/A	$P = 1.0, R =$ N/A
Stays	$P = 1.0, R = .97$	$P = 1.0, R = 1.0$
Speed: Preprocessing	0s	1.6s
Speed: LDA Execution	5h51m	31m37s
Scenario 2		
Spikes	$P = .17, R = 1.0$	$P = 1.0, R = 1.0$
Drops	$P = .20, R = 1.0$	$P = 1.0, R = 1.0$
Stays	$P = 1.0, R = .96$	$P = 1.0, R = 1.0$
Speed: Preprocessing	0s	1.6s
Speed: LDA Execution	5h52m	32m17s
Scenario 3		
Spikes	$P = .66, R = .66$	$P = .75, R = 1.0$
Drops	$P = .33, R = 1.0$	$P = 1.0, R = 1.0$
Stays	$P = 1.0, R = .96$	$P = 1.0, R = .99$
Speed: Preprocessing	0s	1.6s
Speed: LDA Execution	5h50m	37m37s

Table 4: Results of the study on simulated data.

5.4 Real World Case Studies

We now apply both approaches to two open source software projects, PostgreSQL and JHotDraw, to evaluate the performance of both approaches on real-word software projects.

For each case study, we calculate the precision metric by randomly sampling the set of events discovered by each approach. As we must verify each event manually, we select just enough samples to yield a 90% confidence level with a margin of error of 10%. Our sample size, n, for each project and each approach is calculated as $n = \left(\frac{z_{\alpha/2}\sqrt{p(1-p)}}{B}\right)^2$ where $B = 0.10$ and $1 - \alpha = 0.90$, so $z_{0.05} = 1.645$ [23]. Since we have no prior knowledge on the probability p of each event, we set p to 0.5 and thus our sample size n is 68. As we do no have full knowledge of every possible event in these case studies, we can not compute the recall measure.

We developed a tool that, for each software project, randomly selects any number of discovered events. For each event selected, the tool presents the words for the topic associated with the event, the version at which the event occurred, and the top 10 documents associated with the topic. Armed with this information, the first author examined the project documentation, looking for evidence that supports the change.

	Hall	**DiffLDA**
PostgreSQL		
Spikes and Drops	$P = .62 \pm .10$	$P = .90 \pm .10$
Speed: Preprocessing	0s	3m12s
Speed: LDA Execution	27h40m	13h25m
JHotDraw		
Spikes and Drops	$P = .69 \pm .10$	$P = .92 \pm .10$
Speed: Preprocessing	0s	3.9s
Speed: LDA Execution	10h28m	6h22m

Table 5: Results of the two case studies.

In order to reduce bias, during the manual verification of each event, the first author was unaware which approach produced the event. It is not until after the verification is complete that we map the analyzed events back to each approach.

Finally, in the analysis of the case studies, we do not consider stay events, as they are most difficult to manually validate: the lack of a topic in documentation could very likely be due to a lack in the project's documentation process, as opposed to an actual stay in the source code. We also do not differentiate between spikes and drops, since an entry in the documentation such as "cleaned up regular expression code" could just as likely indicate an increase or a decrease in the metric value.

Results for PostgreSQL: PostgreSQL is a large open source database system, written in the C programming language (`http://www.postgresql.org`). We study 50 release versions (7.0–8.3.5) containing a total of 31,769 source code documents over 8 years (2000–2008). We model the history of PostgreSQL using $T = 100$ topics.

Table 5 contains the values for precision and speed. We found that the Hall approach achieved a precision of 62% with an execution time of over 27 hours, while DiffLDA was able to improve the precision by 47% to 90%, while only requiring 50% of the execution time (about 13.5 hours).

To illustrate the topic evolutions discovered by each approach, we show two examples in Figures 5(a) and 5(b). Figure 5(a) shows an example of a discovered topic evolution that exhibited similar behavior in the two approaches, the *word stemming* topic, which involves reducing queries and string data to their root form. The figure indicates that the evolutions discovered by the two approaches are quite similar. However, Figure 5(b) shows the evolution of the *joining relations* topic, which is responsible for implementing the `SQL JOIN` clause on one or more relations. Contrary to the previous example, these two evolutions appear to be very dissimilar in shape, trends, and magnitudes, even though the evolutions represent the same topic.

Results for JHotDraw: JHotDraw is a medium-sized, open source, 2-D drawing framework developed in the Java programming language as an exercise of good program design (`http://www.jhotdraw.org`). We study 12 public release versions (5.2.0–7.4.1) containing 5,220 source code documents over 6 years (2000–2006). We model the history of JHotDraw with $T = 46$ topics.

Table 5 lists the values for precision and speed for the two approaches. Again, DiffLDA outperforms the Hall method in both precision and speed, with improvements of 33% and 38%, respectively, over the Hall approach.

To illustrate the topic evolutions discovered by each approach, we show two examples in Figures 5(c) and 5(d). Figure 5(c) shows an example of a discovered topic evolution that exhibited similar behavior in the two approaches: the *undo* topic, which implements the Undo design pattern [24]. The figure indicates that the evolutions discovered by the two approaches are extremely similar. However, Figure 5(b) shows the evolutions of the *drag and drop* topic, which is responsible for implementing the drag and drop interface functionality. Contrary to the previous example, these two evolutions appear to be very dissimilar, disagreeing when and how the topic evolved.

DiffLDA improved precision by 33–47% and reduced runtime by 38–50% relative to the Hall approach.

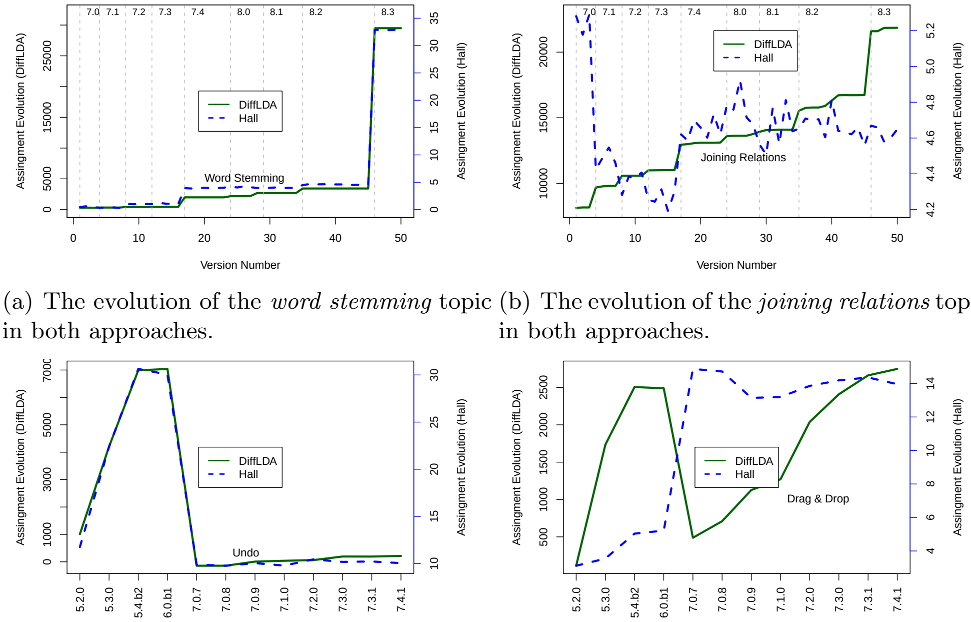
6 Discussion and Threats to Validity

6.1 Limitations of DiffLDA

When DiffLDA compares two versions of a source code document, the edits are saved into the delta documents. Although this achieves our goal of eliminating duplication and capturing the edits to the documents, we sacrifice the context of the original documents: the words in the delta documents are no longer contextualized by surrounding words and functions from the original source code document. Because of this, DiffLDA works well for context-free (i.e., unigram bag of words) topic models, such as LDA. It would be difficult, but interesting to extend our approach to an N-gram model, which has been shown to be a promising technique for topic modeling [25].

6.2 Limitations of Evaluation Techniques

As was mentioned previously, there is currently no benchmark data set for topic evolution mining. As such, researchers (including ourselves) can not compare their results to known standards, and instead must rely on ad hoc analyses that may not fully capture the strengths and weakness of the approach under consideration.



(a) The evolution of the *word stemming* topic in both approaches.

(b) The evolution of the *joining relations* topic in both approaches.

(c) The evolution of the *undo* topic in both approaches.

(d) The evolution of the *drag and drop* topic in both approaches.

Figure 5: Example topic evolutions from the two case studies. Subfigures (a) and (b) are produced from the PostgreSQL project, while (c) and (d) are produced from the JHotDraw project.

In our specific case, our precision metric is not perfect. First, it is based on human judgment as to whether a change in the evolution was justified or not, based on corresponding documentation entries. However, the documentation could be lacking or out of date, or the human could misinterpret the topic or the documentation. Second, even if a change in the evolution is corroborated by documentation, it is not clear whether the magnitude of the change is justified—documentation often does not indicate how many lines of code were involved in a change to a topic's implementation.

6.3 Generality of Results

Although we studied two projects from different domains, of different sizes, and implemented in different programming languages, we cannot necessarily generalize our results to all other projects. First, the projects we studied were both open source, and therefore we cannot generalize our results to closed source projects developed in the industry. Second, both projects we studied followed fairly rigorous variable naming schemes and often used descriptive comments where possible, which allowed the LDA model to work. Should a project not follow standard naming schemes or have common commenting practices, topic modeling would not be an effective tool.

7 Conclusion

Probabilistic topic modeling techniques, such as LDA, have the unstated assumption that the documents in the corpus are not repeated, i.e., that each document in the corpus is unique. We have shown that version histories of source code strongly violate this assumption, as most versions incrementally change less than 7% of the files in the systems. We also showed that LDA produced unexpected results when documents are repeated—topics were less general, and as a result, documents were described by fewer topics and no longer possessed a high similarity measure to other documents.

These results motivated us to propose a new approach for handling source code histories, called DiffLDA, which explicitly captures only the changes between document versions, effectively eliminating the duplication that caused LDA to behave unexpectedly. Such an approach results in a fine-grained representation of topic evolution, as it models the change of each topic from version to version at the word granularity. We evaluated our approach on simulated and real-world data and compared our approach to the traditional LDA approach. We found that DiffLDA increases precision performance as much as 47% as well as reduces execution time by up to 50%.

References

[1] D. Blei and J. Lafferty, "Topic models," *Text Mining: Theory and Applications. Taylor and Francis, London, UK*, 2009.

[2] D. Blei, A. Ng, and M. Jordan, "Latent dirichlet allocation," *The Journal of Machine Learning Research*, vol. 3, pp. 993–1022, 2003.

[3] G. Maskeri, S. Sarkar, and K. Heafield, "Mining business topics in source code using latent dirichlet allocation," in *Proc. of the 1st India Software Engineering Conf.* ACM, 2008, pp. 113–120.

[4] P. F. Baldi, C. V. Lopes, E. J. Linstead, and S. K. Bajracharya, "A theory of aspects as latent topics," *SIGPLAN Not.*, vol. 43, no. 10, pp. 543–562, 2008.

[5] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American society for information science*, vol. 41, no. 6, pp. 391–407, 1990.

[6] A. Kuhn, S. Ducasse, and T. Gírba, “Semantic clustering: Identifying topics in source code,” *Information and Software Technology*, vol. 49, no. 3, pp. 230–243, 2007.

[7] D. Blei and J. Lafferty, “Dynamic topic models,” in *Proc. of the 23rd Intl. Conf. on Machine Learning.* ACM, 2006, p. 120.

[8] X. Wang and A. McCallum, “Topics over time: a non-Markov continuous-time model of topical trends,” in *Proc. of the 12th Intl. Conf. on Knowledge Discovery and Data Mining.* ACM, 2006, p. 433.

[9] D. Hall, D. Jurafsky, and C. Manning, “Studying the history of ideas using topic models,” in *Proc. of the Conf. on Empirical Methods in Natural Language Processing.* Association for Computational Linguistics, 2008, pp. 363–371.

[10] E. Linstead, C. Lopes, and P. Baldi, “An Application of Latent Dirichlet Allocation to Analyzing Software Evolution,” in *Proc. of the 7th Intl. Conf. on Machine Learning and Applications.* IEEE Computer Society, 2008, pp. 813–818.

[11] T. Cover and J. Thomas, *Elements of information theory.* Wiley, 2006.

[12] 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 *Proc. of the 2008 IEEE Intl. Conf. on Data Mining.* IEEE Computer Society, 2008, pp. 3–12.

[13] T. Griffiths and M. Steyvers, “Finding scientific topics,” *Proc. of the National Academy of Sciences*, vol. 101, p. 5228, 2004.

[14] H. M. Wallach, “Structured topic models for language,” Ph.D. dissertation, University of Cambridge, 2008.

[15] Y. Teh, M. Jordan, M. Beal, and D. Blei, “Hierarchical dirichlet processes,” *Journal of the American Statistical Association*, vol. 101, no. 476, pp. 1566–1581, 2006.

[16] H. Jiang, T. Nguyen, I. Chen, H. Jaygarl, and C. Chang, “Incremental latent semantic indexing for effective, automatic traceability link evolution management,” *Proc. of 23rd IEEE/ACM ASE*, vol. 8.

[17] D. Poshyvanyk and A. Marcus, “Combining formal concept analysis with information retrieval for concept location in source code,” in *Proc. of the 15th IEEE Intl. Conf. on Program Comprehension.* IEEE Computer Society, 2007, pp. 37–48.

[18] M. P. Robillard and G. C. Murphy, "Concern graphs: finding and describing concerns using structural program dependencies," in *Proc. of Intl. Conf. on Software Engineering*, 2002, pp. 406–416.

[19] A. Hindle, M. W. Godfrey, and R. C. Holt, "What's hot and what's not: Windowed developer topic analysis," in *Proc. of the 25th IEEE Intl. Conf. on Software Maintenance.* IEEE, September 2009, pp. 339–348.

[20] J. Hintze and R. Nelson, "Violin Plots: A Box Plot-Density Trace Synergism." *The American Statistician*, vol. 52, no. 2, 1998.

[21] S. Bird, R. Dale, B. Dorr, B. Gibson, M. Joseph, M. Kan, D. Lee, B. Powley, D. Radev, and Y. Tan, "The ACL anthology reference corpus: A reference dataset for bibliographic research in computational linguistics," in *Proc. of the 6th Intl. Conf. on Language Resources and Evaluation Conference*, 2008, pp. 1755–1759.

[22] A. K. McCallum, "Mallet: A machine learning for language toolkit," 2002, http://mallet.cs.umass.edu.

[23] R. Scheaffer and J. McClave, *Probability and statistics for engineers.* Duxbury Press Boston, Massachusetts, USA, 1995.

[24] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software.* Addison-Wesley, Reading, MA, 1995.

[25] X. Wang, A. McCallum, and X. Wei, "Topical n-grams: Phrase and topic discovery, with an application to information retrieval," in *Proc. of the 7th IEEE Intl. Conf. on Data Mining*, 2007, pp. 697–702.