Modeling the Evolution of Topics in Source Code Histories

Stephen W. Thomas, Bram Adams, Ahmed E. Hassan, Dorothea Blostein
Software Analysis and Intelligence Lab (SAIL)
School of Computing, Queen's University, Kingston, ON, Canada
{sthomas, bram, ahmed, blostein}@cs.queensu.ca

ABSTRACT

Studying the evolution of topics (collections of co-occurring words) in a software project is an emerging technique to automatically shed light on how the project is changing over time: which topics are becoming more actively developed, which ones are dying down, or which topics are lately more error-prone and hence require more testing. Existing techniques for modeling the evolution of topics in software projects suffer from issues of data duplication, i.e., when the repository contains multiple copies of the same document, as is the case in source code histories. To address this issue, we propose the *Diff* model, which applies a topic model only to the changes of the documents in each version instead of to the whole document at each version. A comparative study with a state-of-the-art topic evolution model shows that the Diff model can detect more distinct topics as well as more sensitive and accurate topic evolutions, which are both useful for analyzing source code histories.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics

General Terms

Experimentation, Measurements

Keywords

Software evolution, topic models, software evolution, LDA

1. INTRODUCTION

Software developers rely on historical software repositories (e.g., source code, mailing lists, and commit logs) for understanding their projects and making maintenance decisions. For example, developers look for the main business concepts embedded in the source code [20]; they browse mailing lists to find discussions about collaborative design decisions [25, 30]; and they read commit logs to see what has recently changed in the source code [13]. However, since these repositories are often large and have no explicit organization, they can be difficult to browse and understand.

MSR'11, May 21–22, 2011, Waikiki, Honolulu, HI, USA

Topic models, such as latent Dirichlet allocation (LDA) [6], have recently been used to help developers understand their repositories [1, 2, 7, 13, 15, 18–20, 23, 28]. Topic models are statistical methods that automatically extract a set of topics from a repository; *topics* are collections of words that co-occur frequently in the repository. Each document is assigned a small set of topics that describe it; a document might be assigned topics like "payroll processing" and "mouse events". The main benefit is that documents can now be linked through the topics they contain; all documents containing the topic *"payroll processing"* can easily be grouped together, no matter to which physical package they belong.

Topic evolution models, which describe how topics change over time, also have the potential to help software developers. For example, topic evolution models can indicate when particular topics were added, modified, or deleted from the source code; the "payroll processing" topic was added in version 3 and saw additional growth in version 7. Developers can also learn which topics are becoming more *scattered* across the system [8,16]; the "payroll processing" topic is now in 100 documents, but it began in just two. Since topics are closely related to concerns and features [2], topic models can provide a deep understanding of the project's history that is difficult to obtain from reading the source code alone.

The state-of-the-art topic evolution model, called the Hall model [12], was initially designed for corpora that have different properties than source code histories. The Hall model typically operates on corpora in which each version is completely different (e.g., conference proceedings) from the previous. Files in source code histories, however, are typically *incrementally updated* between versions (e.g., `main.c` version 34 versus `main.c` version 35), resulting in partial or full duplications of files between successive versions. After applying the Hall model to source code histories, we have found that this *duplication effect* causes unintended results that reduces its effectiveness.

To combat these issues, we propose the *Diff* model, an extension to the Hall model that operates only on the changes between versions of the repository. Through case studies on simulated data, and the JHotDraw and PostgreSQL projects, we show that the Diff model is able to produce more distinct topics and more sensitive and accurate topic evolutions when applied to source code histories, as compared to the Hall model. This paper makes the following contributions:

– We present and formalize the Diff model, which alleviates the data duplication issue.
– We empirically compare the Hall and Diff models through case studies on simulated data and open-source projects.

The data and results of our work are publicly available [26].

Label	Top words	Top 3 matching classes
file filter	*file uri chooser urichoos save filter set jfile open*	`JFileURIChooser, URIUtil, AbstractSaveUnsavedChangesAction`
tool bar	*editor add tool draw action button bar view creat*	`DrawingPanel, ODGDrawingPanel, PertPanel`
undoabl edit	*edit action undo chang undoabl event overrid*	`NonUndoableEdit, CompositeEdit, UndoRedoManager`
connect figur	*figur connector connect start end decor set handl*	`ConnectionStartHandle, ConnectionEndHandle, Connector`
bezier path	*path bezier node index mask point geom pointd*	`CurvedLiner, BezierFigure, ElbowLiner`

Table 1: Example topics from JHotDraw source code version 7.5.1.

2. TOPIC MODELS

Topic models are the result of any method that automatically extracts topics from a corpus of text documents [5], where *topics* are collections of words that co-occur frequently in the corpus. Due to the nature of language usage, the frequently co-occurring words constituting a topic are often semantically related—for example, one might find the words {*user, account, password, authentication*} in one topic, and {*button, click, mouse, drag, drop*} in another. Documents can be structured by the topics they contain, and the entire corpus can be organized in terms of these discovered topics.

Topic models are a useful tool for exploring and understanding the unstructured text found in many software repositories. They have recently been applied to various software engineering problems, such as concern location [2, 17, 20], traceability link recovery [1], trend analysis in commit logs [13], security trend analysis [23], bug localization [19], bug prediction via new coupling/cohesion metrics [9,18], and source code evolution [16, 28].

LDA is a popular probabilistic topic model [6]. LDA models each document in a corpus as a multi-membership mixture of K topics, and each topic as a multi-membership mixture of the words in the corpus vocabulary. This means that each document can contain more than one topic, and each word in the repository can be part of more than one topic. Hence, LDA is believed to be able to discover a set of ideas or themes that well describe the entire software repository [5].

Table 1 shows example topics discovered by LDA from version 7.5.1 of the source code of JHotDraw [14], a framework for creating simple drawing applications. For each topic, the table shows an automatically-generated two-word topic label, the top (i.e., highest probable) words for the topic, and the top three matching Java classes in JHotDraw. The topics span a range of concepts, from opening files to drawing Bezier paths. The discovered topics intuitively make sense and the top-matching classes match our expectations—there seems to be a natural match between the "Bezier path" topic and the `CurvedLinear` and `BezierFigure` classes.

LDA is completely untrained and automatic—no training data or example topics are required before LDA can discover these semantic themes. LDA uses word co-occurrences in documents to automatically discover topics—topics consist of those words that happen to co-occur frequently throughout the corpus. LDA then evaluates each document to determine which topics best describe it. These properties of LDA—ease and power—are likely the reason it has seen recent success in mining unstructured software repositories.

2.1 Source Code Preprocessing

Before LDA (or any topic model) is applied to source code, several preprocessing steps are generally taken in an effort to reduce noise and improve the resulting topics: characters related to the syntax of the programming language (e.g., "`&&`", "`->`") are removed; programming language keywords (e.g., "`if`", "`while`") are removed; identifier names are split into multiple parts based on common naming conventions (e.g., "`oneTwo`", "`one_two`"); common English-language stopwords (e.g., "the", "it", "on") are removed; word stemming is applied to find the root of each word (e.g., "changing" becomes "chang"); and in some cases, the vocabulary of the resulting corpus is further pruned by removing words that occur in over 80% of the documents or under 2% of the documents.

The main idea behind these steps is to capture the semantics of the developers' intentions, which are thought to be encoded within the identifier names and comments in the source code. The rest of the source code (i.e., special syntax, language keywords, and stopwords) are just noise and will not be beneficial to the results of topic models.

2.2 Topic Metrics

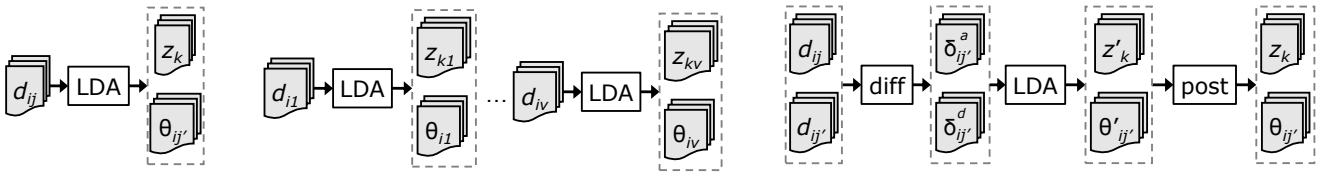
The output of LDA can be post-processed to compute metrics on the discovered topics [2,28]. For example, metrics can measure the similarities between topics (based on shared words) and similarities between documents (based on shared topics). These similarities can help in refactoring, program understanding, and other maintenance tasks, allowing developers to browse the source code from a topic (semantic) view, as opposed to the traditional package (physical) view.

In this paper we focus on the assignment metric of topics, although the models we study generalize to any metric. The ***assignment*** of a topic is defined as the summation of the membership of all documents for that topic. The assignment metric gives a good indication of the total weight or volume of a topic throughout the repository. For example, if the topic with top words {*parsing grammar rule sentence*} has a higher assignment than the topic with top words {*mouse click right left move*}, then we can infer that the repository has more lines of code about parsing than it does about mouse events.

3. TOPIC EVOLUTION MODELS

A *topic evolution model* is a topic model that accounts for time in some way, allowing the documents to have timestamps and the corpus to be versioned. Topic evolution models are useful for detecting and analyzing how topics change, or evolve, over the lifetime of a corpus. Some topics might become more and more present, increasing their overall importance. Other topics may die completely, indicating a marked change in the underlying corpus.

Topic evolution models have several applications in software engineering. First, knowing when a topic is first introduced or heavily modified in a repository is useful for program understanding, since it reveals the history of important concepts within the code [16]. Second, measuring the scatter (i.e., amount of spread) or tangle of a topic over time is useful for refactoring efforts, since it identifies potential bad code smells [8]. Third, understanding which topics were modified at a particular version can glean insight into which features or concerns were modified at that version, which is



(a) The Hall model applies LDA to all versions at the same time.

(b) The Link model applies LDA to each version separately, then links the topics across versions.

(c) The Diff model adds a diff step to isolate the changes between versions of each document and a post-processing step to reconstruct the membership vectors. Here, $d_{ij'}$, is the next version of d_{ij}.

Figure 1: A graphical depiction of the three topic evolution models.

useful for maintenance and re-engineering [13]. Topic evolution models thus allow a higher-level semantic analysis of software evolution.

Topic evolution models typically involve two steps. In the first, topics are extracted from the corpus in the usual way for topic models, based on word co-occurrences in the documents. In the second, the amount of change to a topic at each point in time is measured by the topic evolution model, typically by computing a topic metric (like the assignment metric described in Section 2.2) on each individual version and then plotting the metric across all versions.

Many topic evolution models have been proposed [4, 12, 22, 29], but most make assumptions that are too restrictive for use on software repositories. Thus far in the software engineering literature, two topic evolution models have been employed: the *Hall model* and the *Link model*. Before we describe each, it is necessary to introduce some notation.

3.1 Notation

LDA discovers K ***topics***, $z_1, \ldots, z_K$, where K is an input to the model. For each topic z_i, LDA produces an m-length word (or term) membership vector ϕ_{z_i} that describes the probability that each word appears in topic z_i. (There are m unique words in the vocabulary of the corpus.) Additionally, for each ***document*** d_j in the corpus, $j = 1, \ldots, n$, LDA produces a K-dimensional topic membership vector θ_{d_j} that describes the probability that each topic appears in d_j. A document is a general notion in LDA, and can be any string of characters, although it is typically defined to be either an entire source code file or an individual method.

A ***version*** V of a corpus is a set of documents $\{d_1, d_2, ...\}$ with the same time-stamp $t(V)$. The ***version history*** H of a corpus is the set of versions related to the corpus, $H = \{V_1, V_2, \ldots, V_v\}$. We say that d_{ij} is the document with index i in version V_j, and there are $|V_j|$ documents in a particular version V_j of the corpus.

Finally, the ***evolution*** E of a metric m of a topic z_k is a time-indexed vector of metric values for that topic: $E(z_k, m) = [m(z_k, t(V_1)), m(z_k, t(V_2)), ..., m(z_k, t(V_v))]$.

3.2 The Hall Model

The Hall model, proposed by Hall et al. [12] and first used on software repositories by Linstead et al. [16] and Thomas et al. [28], is an intuitively simple topic evolution model that applies LDA to all versions of all documents at once (see Figure 1(a)). Thus, LDA is executed only once on all $|V_1| + \ldots + |V_v|$ documents at the same time.

During the post-processing phase, the topic metrics are computed by first isolating all documents in a particular version. For example, the assignment of a topic z_a at version V_j is computed as the summation of the topic's memberships across all documents in V_j:

$$\mathrm{A}(z_a, V_j) = \sum_{i=1}^{|V_j|} \theta_{d_{ij}}[a]. \quad (1)$$

This intuitive and simple application of LDA to all versions makes the Hall model attractive for use on many datasets.

	Min.	Med.	Mean	Max.
JHotDraw				
% files changed per version	<0.1	33.8	35.9	89.5
% words changed per changed file	<0.1	0.1	0.2	35.2
PostgreSQL				
% files changed per version	<0.1	6.3	15.9	73.8
% words changed per changed file	<0.1	0.1	0.2	17.7

Table 2: Change characteristics of JHotDraw and PostgreSQL.

3.3 The Link Model

The Link model, proposed by Mei et al. [22] and first used on software repositories by Hindle et al. [13], takes a different approach than the Hall model. The Link model applies LDA to each version of the repository *separately*, followed by a post-processing phase to link topics across versions (see Figure 1(b)). Once the topics are linked, the topic evolutions can be computed in the same way as in the Hall model (Equation 1).

If there are v versions of the corpus, then LDA is executed v times on an average of $\frac{1}{v} * (|V_1| + \ldots + |V_v|)$ documents each time. The post-processing phase must iteratively link topics found in V_i to the topics found in V_{i-1}. This process inherently involves the use of similarity thresholds to determine whether two topics are similar enough to be called the same, since LDA is a probabilistic process and it is not guaranteed to find the exact same topics in different versions of a corpus. As a result, at each successive version, some topics are successfully linked while some topics are not, causing past topics to "die" and new topics to be "born". Additionally, it is difficult to allow for gaps in the lifetime of a topic—a death at version V_i and then a rebirth at V_j for some $j > i + 1$.

For these reasons, we leave the evaluation of the Link model to future work, and instead focus the comparison of our proposed model against the more popular Hall model.

4. THE DIFF MODEL

We now motivate and present a new topic evolution model, called the *Diff* model.

4.1 Motivation for the Diff Model

Traditional topic evolution models were designed for corpora in which the documents are unique across time. This is

true, for example, for conference proceedings—it is not the case that an article one year is only slightly updated and republished the next year. Instead, each article (i.e., the specific combination of words within an article) is unique across time. However, source code repositories are typically updated incrementally, with each version making only small changes to the previous. Therefore, we would expect to see significant overlap in the data (i.e., word co-occurrences) between versions.

To quantify the change characteristics in historical source code repositories, we looked at two open source systems, JHotDraw [14] and PostgreSQL [24]. For JHotDraw, we analyzed the source code changes over 13 releases (versions 5.2.0–7.5.1) and for PostgreSQL, we analyzed the source code changes over 46 releases (versions 7.0.0–8.3.5). Table 2 summarizes our findings, and we make two important observations.

1. Most files are not altered between versions. On average, between 16% (PostgreSQL) and 36% (JHotDraw) of the source code files experienced some change between versions, measured by the number of files that had any change activity (i.e., lines added, removed, or modified). In other words, on average at least 64% (JHotDraw) and up to 84% (PostgreSQL) of the source code files are exact duplicates from release to release. These unaltered documents will obviously have the same word co-occurrences as their previous versions, since no changes were made.

2. Most changes are very small. For the average file that experienced a change between versions, only 0.1% (both PostgreSQL and JHotDraw) of its words actually changed, measured as the number of changed words over the number of total words in the file. Almost all of a file's content remains unaltered, and hence the word co-occurrences will largely be the same.

We hypothesize that the above observations affect the results obtained by the Hall model. Recall that the Hall model applies LDA to all versions of all files. Since we know that most files are not changed at all between versions, and even the files that are changed are not changed by much, we can conclude that the word co-occurrences that LDA operates on will be skewed in the direction of the duplicated files.

To illustrate this, Figure 2 shows a simple example. The repository on the left is a normal repository (e.g., conference proceedings) with two unique documents. LDA will discover a topic from the most commonly co-occurring words, in this case {*a, b*}. This is the preferable behavior of LDA. The repository on the right has four versions of two unchanging documents. In this case, LDA will discover two topics ({*a, b, c*} and {*a, b, d*}), instead of discovering one topic ({*a, b*}) as it would have if it were only applied to the first version of the two documents. We call this the *duplication effect*, which occurs (to varying degrees) any time LDA is run on two or more near copies of a document.

4.2 Diff Model Description

In order to address the data duplication effect found in source code histories, we propose a simple but effective topic evolution model, the Diff model. The key idea is that the Diff model prepends a *diff step* to the Hall model to isolate the changes between successive versions of each document. This diff step effectively removes all duplication and leaves only the changed portion of the document, hence ridding the corpus of the duplication effect. The Diff model can

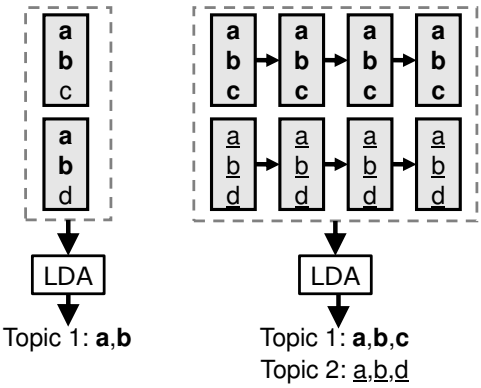


Figure 2: On the left, a repository without duplication (2 unique files, 3 words each) yields a preferable topic. On the right, a repository with duplication (2 files, 4 unchanged versions each) yields confounded topics (should only be a single topic {*a, b*}).

be thought of as an extension to the Hall model, since the models are largely the same. However, as we will show in this paper, the diff step is critical when applying topic evolution models to software repositories with duplication.

Figure 1(c) depicts the Diff model process. For each source code document d_{ij} in the system, we first compute the edits between successive versions V_j and $V_{j'}$ ($j' = j+1$) using the GNU `diff` file comparison tool [10]. `diff` classifies each edit as an *add* or *delete*, depending on whether the edit resulted in more or fewer lines of code, respectively. If an existing line is modified, it is considered by `diff` to be deleted and then added again. For each version of each document, we create two *delta documents*, $\delta^a_{ij'}$ and $\delta^d_{ij'}$. We place all the added lines between d_{ij} and $d_{ij'}$ into $\delta^a_{ij'}$ and all the deleted lines into $\delta^d_{ij'}$. We use the notation $|\delta^a_{ij'}|$ to represent the number of words in $\delta^a_{ij'}$.

We must handle two special cases: 1) When we encounter a new document d_i at version V_j (either because $j = 1$ or d_i is a new document), we classify the entire document as added words, and thus we add the entire document to the delta document δ^a_{ij}; and 2) when a document d_i is removed at version V_j, we classify the entire document as deleted words, and thus add the entire document to the delta document δ^d_{ij}.

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

Finally, we post-process the output of LDA to compute the membership values of the original documents at each point in time. The corresponding $\theta_{d_{ij}}$ vector of a document d_i at version V_j is defined recursively as

$$\theta_{d_{ij}} = \frac{\theta_{d_{i\hat{j}}} \left|d_{i\hat{j}}\right| + \theta_{\delta^a_{ij}} \left|\delta^a_{ij}\right|}{\left|d_{i\hat{j}}\right| + \left|\delta^a_{ij}\right|} - \varphi\left(\frac{\theta_{d_{i\hat{j}}} \left|d_{i\hat{j}}\right| - \theta_{\delta^d_{ij}} \left|\delta^d_{ij}\right|}{\left|d_{i\hat{j}}\right| - \left|\delta^d_{ij}\right|}\right) \quad (2)$$

where $\varphi()$ is the normalizing function and $\hat{j} = j - 1$ is the index of the previous version of document d_i. n() accounts for the scenario when more words matching a given topic were subtracted than were in the previous version of the document for that topic (e.g., subtracting 20 words about topic A when the previous version only had 10 words about topic A), which is unlikely but possible because LDA is a probabilistic process.

The intuition behind Equation 2 is that for each version of a document, we adjust the θ vector by adding the lines of the δ^a document and subtracting the lines of the δ^d document, thus arriving at a representative state of the document at each version. This cumulative definition is necessary since we only model the *changes* at each version, but we want to know the θ vector for the entire document at each version.

4.3 On The Origin of the Diff Model

The Diff model was originally motivated by our own experiences and struggles when applying the Hall model to source code histories. In a previous study, we applied the Hall model to the source code history of JHotDraw to study its evolutions of topics [28]. At the time, the Hall model was the known standard for such a task, and we found the results to be acceptable.

In a subsequent study, we applied the Hall model to a repository with significantly more versions than JHotDraw, and hence, more duplication. Understanding the topic evolutions in this case proved to be difficult, and we felt there was something wrong with the topics and evolutions. After considerable investigation, we hypothesized the cause of the problem to be the duplication effect, and the Diff model was created to address this issue.

We note that the Diff model can be viewed as an extension to the Hall model, for two primary reasons. First, after the diff step, the Hall and Diff models perform equivalent actions on the data. Second, when applied to a traditional corpus with no duplication (i.e., a history of conference proceedings), the diff step will have no effect—it will essentially be a `noop`. Thus, as we will show, the Diff model improves the results of Hall on repositories that have duplication; on all other repositories, the two models are equivalent.

5. EMPIRICAL EVALUATION

We now perform an empirical evaluation of the Diff and Hall models. As was described in Section 4.1, we hypothesize that when the Hall model is applied to a repository with duplication, it will generate imperfect topics that confound multiple concepts. On the other hand, the Diff model will create more distinct topics that stand on their own and thus allow the documents to be described more naturally. In fact, producing distinct topics is known to be a desirable property for topic models [3].

In addition, because the topics discovered by the Diff model are more distinct and better describe the documents, we hypothesize that the resulting topic evolutions will more accurately describe the changes to the repository.

We now formulate the research hypotheses that we focus on in this section.

Hypothesis 1. The removal of data duplication will result in more *distinct* topics.

Hypothesis 2. More distinct topics will allow the discovery of more *sensitive* evolutions.

Hypothesis 3. More distinct topics will allow the discovery of more *accurate* evolutions.

We test our first two hypotheses by conducting case studies on two real-world open source systems (Section 5.1), and we build a *simulated* project, whose properties are well-known, to test our third hypothesis (Section 5.2).

5.1 Evaluation on Real-World Systems

Our goal in this section is to determine whether there is a difference between the Hall and Diff models in the distinctness of the discovered topics and the sensitivity of the discovered topic evolutions.

5.1.1 Studied Systems

We applied both models to the source code histories of two open source systems: JHotDraw [14] and PostgreSQL [24]. JHotDraw is a medium-sized drawing framework implemented in Java and has been the subject of many previous studies. PostgreSQL is a large database management system and is chosen due to its extensive documentation.

Our JHotDraw dataset consists of 13 releases (versions 5.2.0–7.5.1). The latest release contains 613 files totaling 84K source lines of code (SLOC). Our PostgreSQL dataset is comprised of 46 release versions (versions 7.0.0–8.3.5). The latest release contains 844 files totaling 501K SLOC.

5.1.2 Study Setup

We preprocessed the source code of each system using the steps described in Section 2.1. Namely, we isolated identifier names and comments from the source code, discarding special syntax, programming language keywords, and English language stopwords. We split identifiers, stemmed each word, and pruned the vocabulary so that overly common (>80%) or overly rare (<2%) words are removed.

For JHotDraw, the preprocessing resulted in a total of 2.3M words (964 of which are unique) in 5,833 documents, totaling 29MB of data on disk. For PostgreSQL, the preprocessing resulted in 40M words (2867 of which are unique) in 29,559 documents, totaling 299MB on disk.

For the actual LDA computation, we used MALLET version 2.0.6 [21]. We ran for 10,000 sampling iterations, the first 1,000 of which were used for parameter optimization [11]. The scenarios were executed on a machine running Ubuntu 9.10 with a 2.8GHz 16-core CPU and 64Gb of main memory. We modeled JHotDraw with $K = 45$ topics and PostgreSQL with $K = 100$ topics. We chose more topics for PostgreSQL because it has a larger, more complex code base.

5.1.3 Evaluation Measures

A ***change event*** in a topic evolution is an increase (*spike*), decrease (*drop*), or no change in a metric value (*stay*) between successive versions. We classify a change event as a spike or drop if there is at least a 20% increase or decrease in metric value compared to the previous version, 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

$$\text{Event}(m, z_k, V_i) = \begin{cases} \text{spike} & \text{if } c \geq 0.2, \text{or if } m(z_k, V_{i-1}) = 0 \\ & \quad \text{and } m(z_k, V_i) > 0; \\ \text{drop} & \text{if } c \leq -0.2; \\ \text{stay} & \text{otherwise.} \end{cases} \tag{3}$$

A distinct topic is one that stands on its own—it is not similar to any other discovered topics. We define the ***topic distinctness*** of a topic z_i as the mean KL divergence between the word membership vectors of z_i and z_j, $\forall j \neq i$:

$$\text{TD}(\phi_{z_i}) = \frac{1}{K-1} \sum_{j=1, j\neq i}^{K} \text{KL}(\phi_{z_i}, \phi_{z_j}). \tag{4}$$

	Hall	Diff	p-value
JHotDraw			
Mean topic distinctness	3.72	4.43	<0.001
Spikes per version	9.58	15.17	<0.001
PostgreSQL			
Mean topic distinctness	2.56	3.64	<0.001
Spikes per version	2.58	4.49	<0.001

Table 3: Results of our case studies on JHotDraw and PostgreSQL.

Diff topic 63:	*acl role privileg mode oid grant owner roleid*
Diff topic 40:	stmt creat comment defel command defnam
Hall topic 68:	*oid* stmt tupl comment *owner* rel creat *acl list*

Table 4: The "ACL" and "commands" topics from Diff, and the "ACL-commands" topic from Hall. The words in Diff topic 63 are *emphasized* and the words in Diff topic 40 are underlined in the Hall topic 68.

A higher TD measure indicates that a topic is more distinct. We use the TD measure to test Hypothesis 1.

We define the ***evolution sensitivity*** of an evolution $E(z_i)$ as the mean number of detected spikes per version of the system:

$$\text{ES}(E(z_i)) = \frac{|\{\text{Detected spikes and drops in } E(z_k)\}|}{(v-1)}. \quad (5)$$

If a detected evolution has more spikes and drops, then we say it is more sensitive than an evolution with fewer spikes and drops. We use the evolution sensitivity measure to test Hypothesis 2. To ensure that the detected spikes are not false positives, we manually investigate topic evolutions in a controlled environment in Section 5.2.

5.1.4 Results

To illustrate the kind of evolutions that are discovered by the two models, Figures 3(a)–3(c) show three discovered evolutions. Figures 3(a) and 3(c) show examples of evolutions that were similar between the two models for JHotDraw and PostgreSQL, respectively. In these cases, the topics themselves contained similar words and thus their evolutions followed similar paths. Figure 3(b) shows example evolutions from JHotDraw that were different between the two models, despite the topics being similar.

After computing our evaluation measures on the resulting topics and evolutions, we make the following two observations.

Observation 1. The Diff model produces more distinct topics than the Hall model, supporting Hypothesis 1.

Table 3 shows the topic distinction measures for the two models for both JHotDraw and PostgreSQL (Equation 4). In both systems, the Diff model produces significantly more distinct topics, supporting our first research hypothesis.

To illustrate this, consider the following example from the case study on PostgreSQL. The `lockcmds.c` document is responsible for taking a lock command from the user, checking the access control list (ACL) to see if the user has permissions, and performing a lock on a table if the user does have permissions. Thus, the document contains the concepts of "commands" and "access control lists". In the Diff model, this document is matched to two topics: topic 63 (with membership .25), which describes access control lists, and topic 40 (with membership .58), which describes commands. Under the Hall model, `lockcmds.c` is assigned to only one topic, topic 68 (with membership 0.93), which confounds the locking and commands concepts into a single topic. Table 4 shows theses topics and highlights the similarity between Hall's topic 68 and the two Diff topics, 63 and 40. The Hall topic 68 is less distinct, with a topic distinctness of 2.49, compared to the relatively distinct Diff topic distinctnesses of 3.46 (topic 40) and 3.49 (topic 63).

Observation 2. The Diff model is more sensitive to detecting changes in the data than the Hall model, supporting Hypothesis 2.

Table 3 shows the results of evolution sensitivity for the two models for both JHotDraw and PostgreSQL, measured as the mean number of detected spikes per version (Equation 5). In both systems, the Diff model detects significantly more spikes than the Hall model. This finding supports our second hypothesis and is in part a result of Observation 1: since the topics in the Diff model are more distinct, new documents will be matched to more topics, and thus there will be more spikes in the evolution. On the other hand, the Hall model finds less distinct topics (i.e., each topic has multiple concepts confounded), and new documents will tend to be matched to fewer topics, resulting in fewer spikes in the evolution.

To illustrate this difference, consider again the `lockcmds.c` file from PostgreSQL, which was first added to the system at version V_{12} along with a group of similar files. Figure 3(d) shows the evolutions produced by the Diff and Hall models that match the `lockcmds.c` file. Under the Diff model, two topics received spikes between version V_{11} and V_{12}, which were the topics for ACLs and commands. Under the Hall model, however, only a single topic received a spike, which was the confounded ACL and commands topic.

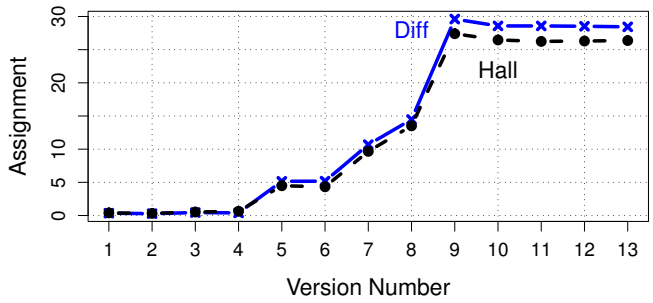
5.2 Evaluation on Simulated Data

We now test Hypothesis 3 by quantifying the *accuracy* of each model when applied to source code histories—whether each model is able to detect source code changes correctly and completely. Hypothesis 2 concluded that the Diff model created more sensitive topic evolutions, but it might be the case that the model is overly sensitive, discovering false-positive change events. To investigate this possibility, we must assess the accuracy of the discovered change events.

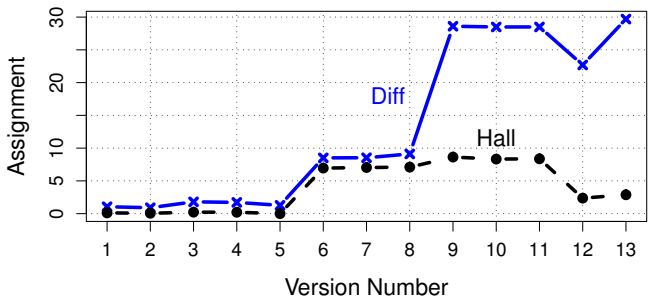
Since there is no public dataset for evaluating the accuracy of topic evolution models, we perform a controlled experiment on a manually-created *simulated* software project. We have created two simple scenarios with a representative variety of source code changes so that we could exactly determine whether the evolutions extracted by the models were accurate— that is, whether the change events detected by each model correspond to the actual changes that we introduced in the source code (precision) and whether the detected evolutions contained all the changes that we introduced in the source code (recall).

5.2.1 Data Generation

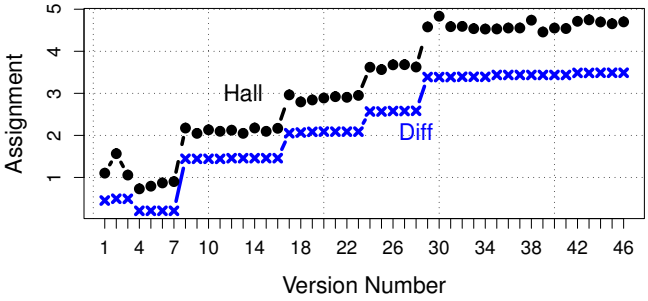
We built the simulated software project by starting with the `backend.access` (version 8.2.1) subsystem of PostgreSQL. The `backend.access` subsystem contains 58 source code files and 8 subdirectories, and is responsible for implementing



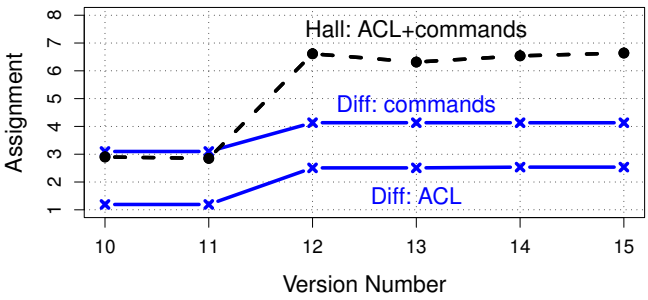
(a) Similar evolutions: the "menu icon" topic.



(b) Dissimilar evolutions: the "color gradient" topic.



(c) Similar evolutions: the "multi xact" topic.



(d) Evolutions for the "ACL" and "commands" topics.

Figure 3: Sample topic evolutions from the case studies on JHotDraw ((a), (b)) and PostgreSQL ((c), (d)). In all plots, the dashed black line (with circles as points) shows the evolution discovered by the Hall model while the solid blue line (with crosses as points) shows the evolution discovered by the Diff model.

functionalities such as hash tables, transactions, and NBTrees. We chose this subsystem due to its medium size and clear functionality definitions.

We created two simulated scenarios as follows. First, we made ten duplicates of all 58 source code files in the `backend.access` to create ten (unchanged) versions of the package. We called these versions (V_1–V_{10}) the baseline scenario.

The first scenario modifies the baseline scenario by introducing three documents from the unrelated `timezone` subsystem at version V_5, then removing all three documents at version V_6. Thus, there are 58 files in versions V_1–V_4, 61 files in version V_5, and again 58 files in versions V_6–V_{10}. This scenario simulates two typical actions: the introduction of new functionality and the removal of existing functionality.

The second scenario starts with the documents of the first scenario and makes the following two additions: 1) Eight documents from the unrelated `ecpg` subsystem were inserted in versions V_9 and V_{10}. The first half of each document was inserted in version V_9, while the second half of each document was inserted at version V_{10}. 2) Five documents from the unrelated `backend.regex` subsystem were inserted in version V_1, they remained (unchanged) in versions V_2 and V_3, and were removed at version V_4.

5.2.2 Study Setup

We preprocessed the source code of each system under study using the steps described in Section 2.1 in the same way as we did for the real-world systems (Section 5.1.2).

For Scenario 1, the preprocessing resulted in a total of 1.2M words (3629 of which are unique) in 583 documents, totaling 9MB of data on disk. For Scenario 2, the preprocessing resulted in a total of 1.2M words (3666 of which are unique) in 614 documents, totaling 9MB on disk.

For the actual LDA computation, we used the same setup as in Section 5.1.2, with the exception that we modeled each simulated scenario using $K = 20$ topics.

5.2.3 Evaluation Measures

To quantify the accuracy of each model, we calculate precision and recall. The ***precision*** of a model describes how many of the discovered change events were correct. We compute the precision of an evolution $E(z_k)$ as

$$P(E(z_k)) = \frac{|\{\text{Correct events in } E(z_k)\}|}{|\{\text{All discovered events in } E(z_k)\}|}. \quad (6)$$

We are able to determine which discovered change events are correct since we have manually created the changes in the simulated project. For example, we expect to see a spike at version V_5 in the evolution relating to the `timezone` subpackage, since we first added the `timezone` documents at V_5.

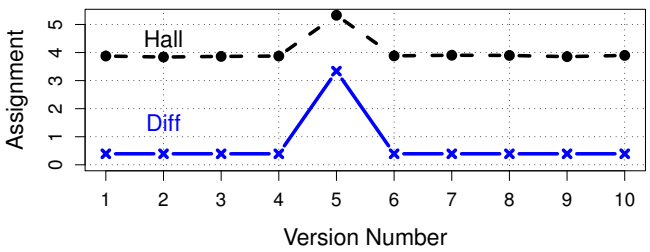
The ***recall*** of a model describes how many of the truth events were discovered by the model. We compute the recall of an evolution $E(z_k)$ as

$$R(E(z_k)) = \frac{|\{\text{Correct events in } E(z_k)\}|}{|\{\text{Truth events}\}|}. \quad (7)$$

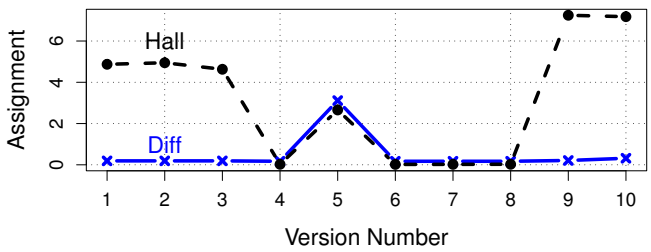
We are able to determine which truth events exist because we manually created the truth events in the source code.

5.2.4 Results

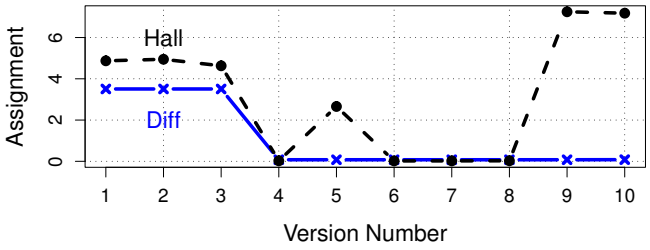
Scenario 1 (Single Change): This scenario contains 10 unchanging versions of 58 documents, with the exception that three documents from the unrelated `timezone` subsystem were inserted at version V_5, and removed at version V_6.



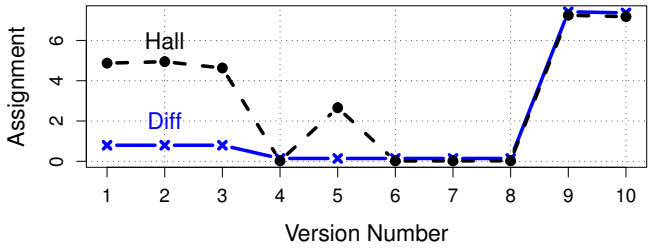
(a) The `timezone` topic evolutions in Scenario 1.



(b) The `timezone` topic evolutions in Scenario 2.



(c) The `regex` topic evolutions in Scenario 2.



(d) The `ecpg` topic evolutions in Scenario 2.

Figure 4: Sample topic evolutions for the simulated scenarios. In all plots, the dashed black line (with circles as points) shows the evolution discovered by the Hall model, while the solid blue line (with crosses as points) shows the evolution discovered by the Diff model.

Expectations: We expect to see a spike in a `timezone` related topic at version 5, a drop at version 6, and no change in assignment to any other topic at any other time.

Results: Figure 4(a) shows the discovered topic evolutions for the `timezone` related topics. (Appendix A shows the actual topics.) In this scenario, the Diff model creates a topic just for these `timezone` documents—the evolution has a value of 0 at all versions except at version V_5, where it spikes to an assignment value of around 3.0, then drops again at version V_6. Indeed, all three of the `timezone` documents have high memberships in this topic, and low memberships in all other topics. Likewise, no documents besides the three `timezone` documents have a non-zero membership in this topic.

The Hall model, on the other hand, does not create a topic solely for the `timezone` documents. Instead, the Hall model assigns the three `timezone` documents to an existing topic that already had a non-zero assignment value from other, unrelated documents. This topic spikes and drops at versions V_5 and V_6, respectively. In this case, we say that the discovered evolution is incorrect, because the change events are discovered in a topic that is not related to the timezone documents.

Table 5 lists the precision and recall measures for Scenario 1 across all topics. In this scenario, there is only one truth spike and one truth drop. Since the Hall model found two incorrect events and no correct events, it achieves a precision and recall measure of 0.0. The Diff model, on the other hand, correctly discovered the truth spike and drop without spurious events, achieving precision and recall values of 1.0. In all cases, the Diff model achieves higher recall and precision measures, and Hypothesis 3 is supported.

Scenario 2 (Multiple Changes): This scenario is the same as Scenario 2 except that eight documents from the `ecpg` subsystem are added at version V_9 and five documents from the `backend.regex` subsystem are present (and unchanging) in versions V_1–V_3.

	Hall	Diff
Simulated Project - Scenario 1 (Single Change)		
Spikes	$P = 0.0, R = 0.0$	$P = 1.0, R = 1.0$
Drops	$P = 0.0, R = 0.0$	$P = 1.0, R = 1.0$
Simulated Project - Scenario 2 (Multiple Changes)		
Spikes	$P = .60, R = 1.0$	$P = .75, R = 1.0$
Drops	$P = .66, R = 1.0$	$P = .66, R = 1.0$

Table 5: Precision (P) and recall (R) results of the experiments on the simulated data.

Expectations: We expect to see a spike for the `timezone` related topic at version V_5, followed by a drop at version V_6. We also expect to see a spike for the `ecpg` related topic at version V_9 and a drop in the `backend.regex` related topic at version V_4.

Results: Figures 4(b)–4(d) show the discovered topic evolutions for the three subpackages involved in the scenario. (Appendix A shows the actual topics.) The Hall evolutions in the three figures are actually showing the same topic, because only a single topic was created to house the `timezone`, `ecpg`, and `backend.regex` related documents. While the three figures show that the expected events were indeed discovered by the Hall method (i.e., a drop at version V_4, a spike and drop at version V_5 and V_6, and a spike at version V_9), the topic itself confounds three separate concepts and hence is not easy to interpret (further supporting Hypothesis 1).

The Diff model, on the other hand, captured all three subpackages in their own distinct topics, each having the expected change events.

Table 5 shows the precision and recall results. In this case, since the Hall model created a new topic for the newly

introduced documents, and captured all of the expected change events correctly, we say that all of the evolutions in Figures 4(b)–4(d) are correct. However, other topic evolutions discovered spurious change events, causing the precision score to be less than 1. The Diff model also discovered a few spurious change events (including the drop shown at version V_4 in Figure 4(d)), and also receives a precision score less than 1. Even still, Hypothesis 3 is supported because the Diff model outperforms the Hall model.

5.3 Further Evaluation of Accuracy

Our case studies on real-world systems have confirmed our first two hypotheses: the Diff model results in more distinct, understandable topics, and it is more sensitive for detecting the changes in a source code repository. Furthermore, our study on a simple, manually-created simulated project provides support for our third hypothesis—that the Diff model produces evolutions that are more accurate.

These initial results, based on mathematical metrics and simulated data, are encouraging. We plan to further test our hypotheses by conducting a user study to determine the usefulness of the models for practitioners. We have already performed an initial user study to this effect, where the first author blindly rated events (spikes and drops) in the topic evolutions of JHotDraw and PostgreSQL against project documentation to determine whether the events were justified [27]. The results of this initial study were positive, with precision scores of 92% for the Diff model, compared to 69% for the Hall model. We are currently preparing a larger scale user study to confirm these findings.

6. LIMITATIONS

Parameter Choices. During our studies, we had to choose values for several model parameters, including the number of topics (K); the delta threshold for classifying an event as a spike or drop; the number of sampling iterations to run in LDA; and the pruning parameters for preprocessing the source code histories. Our choices are somewhat subjective, as there is no standard way to determine optimal values. However, we used the same parameter values in both the Hall and Diff models, providing an opportunity for an equal comparison.

Generality of Results. Although we studied two real-world systems from different domains, of different sizes, and implemented in different programming languages, we cannot necessarily generalize our results to all other systems. First, the systems we studied were both open source, and therefore we cannot generalize our results to proprietary systems developed in the industry. Second, both studied systems follow established variable naming schemes and often use descriptive comments where possible, which permit meaningful topics to be discovered. In projects that do not follow standard naming schemes or have consistent commenting practices, topic models might not be effective tools.

7. CONCLUSION

Topic evolution models have the potential to help software developers understand the histories and trends of their software repositories in a new and deep way. However, traditional topic evolution models were designed for versioned corpora whose documents are never duplicated, such as conference proceedings and newspaper articles. We have found that source code histories deviate from this norm: in some cases, 99% of the source code files in a new version are unmodified copies from the previous version. This *duplication effect* alters the results of topic models.

These characteristics of software repositories motivated us to propose a new topic evolution model, which is a simple but powerful extension to the Hall model. Our model operates only on the changes between versions of the repository, effectively eliminating the duplication effect. We evaluated the Diff model through case studies on two real-world software systems, JHotDraw and PostgreSQL, as well as a simulated software project that is well-understood. We found that the Diff model produces significantly more distinct topics, which are preferable to the confounded topics found by the Hall model. We also found that the more distinct topics allowed more sensitive and accurate evolutions to be discovered, which better model the changes in a source code history.

These encouraging results motivate us to consider additional case studies (simulated and real) to confirm the results seen here, as well as evaluate the Diff model against the Link model.

Acknowledgements. We would like to thank the reviewers for their insightful feedback that helped improve the quality of this paper. This work is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and Research in Motion (RIM).

8. REFERENCES

[1] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor. Software traceability with topic modeling. In *Proc. 32nd Intl. Conf. on Soft. Eng.*, pages 95–104, 2010.

[2] P. F. Baldi, C. V. Lopes, E. J. Linstead, and S. K. Bajracharya. A theory of aspects as latent topics. *ACM SIGPLAN Notices*, 43(10):543–562, 2008.

[3] D. M. Blei, T. L. Griffiths, and M. I. Jordan. The nested chinese resaurant process and bayesian nonparametric inference of topic hierarchies. *J. ACM*, 57(2):1–30, 2010.

[4] D. M. Blei and J. D. Lafferty. Dynamic topic models. In *Proc. 23rd Intl. Conf. on Mach. Learn.*, pages 113–120, 2006.

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

[6] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *J. Mach. Learning Rsrch.*, 3:993–1022, 2003.

[7] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *J. Amer. Soc. for Info. Sci.*, 41(6):391–407, 1990.

[8] M. Eaddy, T. Zimmermann, K. D. Sherwood, V. Garg, G. C. Murphy, N. Nagappan, and A. V. Aho. Do crosscutting concerns cause defects? *IEEE Trans. on Soft. Eng.*, pages 497–515, 2008.

[9] M. Gethers and D. Poshyvanyk. Using relational topic models to capture coupling among classes in Object-Oriented software systems. In *Proc. 26th Intl. Conf. on Soft. Maint.*, 2010.

[10] GNU Diffutils. `http://www.gnu.org/software/diffutils`, 2010.

[11] T. L. Griffiths and M. Steyvers. Finding scientific topics. *Proc. Ntl. Ac. of Sci.*, 101:5228–5235, 2004.

[12] D. Hall, D. Jurafsky, and C. D. Manning. Studying the history of ideas using topic models. In *Proc. Conf. on Emp. Meth. in Nat. Lang. Process.*, pages 363–371. ACL, 2008.

[13] A. Hindle, M. W. Godfrey, and R. C. Holt. What's hot and what's not: Windowed developer topic analysis. In *Proc. 25th Intl. Conf. on Soft. Maint.*, pages 339–348, 2009.

[14] JHotDraw. `http://www.jhotdraw.org`, 2007.

[15] A. Kuhn, S. Ducasse, and T. Girba. Semantic clustering: Identifying topics in source code. *Info. and Soft. Tech.*, 49(3):230–243, 2007.

[16] E. Linstead, C. Lopes, and P. Baldi. An application of latent dirichlet allocation to analyzing software evolution. In *Proc. 7th Intl. Conf. on Mach. Learn. and App.*, pages 813–818, 2008.

[17] E. Linstead, P. Rigor, S. Bajracharya, C. Lopes, and P. Baldi. Mining eclipse developer contributions via author-topic models. In *Proc. 4th Intl. Work. on Min. Soft. Rep.*, pages 30–33, 2007.

[18] Y. Liu, D. Poshyvanyk, R. Ferenc, T. Gyimothy, and N. Chrisochoides. Modeling class cohesion as mixtures of latent topics. In *Proc. 25th Intl. Conf. on Soft. Maint.*, pages 233–242, 2009.

[19] S. K. Lukins, N. A. Kraft, and L. H. Etzkorn. Bug localization using latent dirichlet allocation. *Info. and Soft. Tech.*, 52(9):972–990, 2010.

[20] G. Maskeri, S. Sarkar, and K. Heafield. Mining business topics in source code using latent dirichlet allocation. In *Proc. 1st India Soft. Eng. Conf.*, pages 113–120, 2008.

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

[22] Q. Mei and C. X. Zhai. Discovering evolutionary theme patterns from text: an exploration of temporal text mining. In *Proc. 11th Intl. Conf. on Know. Disc. in Data Min.*, pages 198–207, 2005.

[23] S. Neuhaus and T. Zimmermann. Security trend analysis with CVE topic models. In *Proc. 21st Intl. Symp. on Soft. Rel. Eng.*, pages 111–120, 2010.

[24] PostgreSQL. `http://www.postgresql.org`, 2010.

[25] P. C. Rigby and A. E. Hassan. What can OSS mailing lists tell us? a preliminary psychometric text analysis of the apache developer mailing list. In *Proc. 4th Intl. Work. on Min. Soft. Rep.*, page 23, 2007.

[26] S. W. Thomas. `http://research.cs.queensu.ca/~sthomas`, 2010.

[27] S. W. Thomas, B. Adams, A. E. Hassan, and D. Blostein. DiffLDA: topic evolution in software projects. Technical Report 2010-574, School of Computing, Queen's University, 2010.

[28] S. W. Thomas, B. Adams, A. E. Hassan, and D. Blostein. Validating the use of topic models for software evolution. In *Proc. 10th Intl. Work. Conf. on Src. Code Anal. and Manip.*, pages 55–64, 2010.

[29] X. Wang and A. McCallum. Topics over time: a non-Markov continuous-time model of topical trends. In *Proc. 12th Intl. Conf. on Know. Disc. and Data Min.*, pages 424–433, 2006.

[30] P. Weissgerber, D. Neu, and S. Diehl. Small patches get in! In *Proc. Intl. Work. Conf. on Min. Soft. Rep.*, volume 2008, pages 67–75, 2008.

APPENDIX

A. SELECTED TOPICS

Scenario 1: Diff: Topic 7	
Label	seq search
Words	*time year case continu const offset type rule hentri*
Docs	`scheck.c`, `strftime.c`, `localtime.c`
Scenario 1: Hall: Topic 17	
Label	normal transact
Words	*transact clog xid accum entri statu subtran commit datum*
Docs	`reloptions.c`, `transam.c`, `ginbulk.c`
Scenario 2: Diff: Topic 2	
Label	offset posix section
Words	*time case year continu type tmp offset const result*
Docs	`scheck.c`, `strftime.c`, `localtime.c`
Scenario 2: Hall: Topic 9	
Label	ecpglog ecpgget
Words	*var case chr state struct break reg end assert*
Docs	`scheck.c`, `strftime.c`, `localtime.c`
Scenario 2: Diff: Topic 5	
Label	case sql
Words	*case sqlca ecpg break connect ecpgt var lineno pval*
Docs	`error.c`, `data.c`, `descriptor.c`
Scenario 2: Diff: Topic 14	
Label	scan consist fine
Words	*chr state reg assert end var struct begin subr*
Docs	`regexec.c`, `regcomp.c`, `rege_dfa.c`
JHotDraw: Diff: Topic 6	
Label	icon icon
Words	*icon descriptor bean properti event gen method color set*
Docs	`JAttributeSliderBeanInfo`, `ODGPropertiesPanelBeanInfo`, `AbstractToolBarBeanInfo`
JHotDraw: Diff: Topic 12	
Label	color color
Words	*color gradient space index compon rgb model system min*
Docs	`ColorSystem`, `HSLRGBColorSystem`, `HSLRYBColorSystem`
JHotDraw: Hall: Topic 23	
Label	icon icon
Words	*icon descriptor bean properti event gen method color set*
Docs	`JAttributeTextAreaBeanInfo`, `JDisclosureToolBarBeanInfo`, `JLifeFormattedTexAreaBeanInfo`
JHotDraw: Hall: Topic 5	
Label	stop color
Words	*color gradient space paint focu fraction stop arrai linear*
Docs	`LinearGradientPaintContext`, `MultipleGradientPaintContext`, `MultipleGradientPaint`
JHotDraw: Hall: Topic 14	
Label	color chooser
Words	*color compon slider index icon model space system rgb*
Docs	`AbstractHarmonicRule`, `HSLRGBColorSystem`, `HSLRYBColorSystem`
PostgreSQL: Diff: Topic 37	
Label	multi xact
Words	*page multi xact clog share lock offset ctl xid*
Docs	`slru.c`, `clog.c`, `varsup.c`
PostgreSQL: Diff: Topic 63	
Label	acl acl
Words	*acl role privileg priv mode oid user grant aclcheck*
Docs	`acl.c`, `aclchk.c`, `superuser.c`
PostgreSQL: Hall: Topic 5	
Label	arg lappend
Words	*constraint stmt column list tabl attr index pstate type*
Docs	`analyze.c`, `tablecmds.c`, `tupdesc.c`
PostgreSQL: Hall: Topic 68	
Label	gettext noop
Words	*guc gettext pgc val config conf variabl noop sourc*
Docs	`guc.c`, `variable.c`, `guc-file.c`