Validating the Use of Topic Models for Software Evolution

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

Abstract

Topics are collections of words that co-occur frequently in a text corpus. Topics have been found to be effective tools for describing the major themes spanning a corpus. Using such topics to describe the evolution of a software system's source code promises to be extremely useful for development tasks such as maintenance and re-engineering. However, no one has yet examined whether these automatically discovered topics accurately describe the evolution of source code, and thus it is not clear whether topic models are a suitable tool for this task.

In this paper, we take a first step towards determining the suitability of topic models in the analysis of software evolution by performing a qualitative case study on 12 releases of JHotDraw, a well studied and documented system. We define and compute various metrics on the identified topics and manually investigate how the metrics evolve over time. We find that topic evolutions are characterizable through spikes and drops in their metric values, and that the large majority of these spikes and drops are indeed caused by actual change activity in the source code. We are thus encouraged by the use of topic models as a tool for analyzing the evolution of software.

1. Introduction

Topics are collections of words that co-occur frequently in a corpus of text. Statistical topic models, such as latent Dirichlet allocation (LDA) [1], are used to automatically discover a set of topics within a corpus; in practice it is found that these topics serve to describe the major themes that span the corpus. Recently, researchers have applied topic models to various aspects of a software project, including source code [2]–[5] and documentation [6], [7]. Thus, these discovered topics provide a means of automatically summarizing and organizing the various contents of a software project.

Understanding how the use of a topic evolves, i.e. changes, in a software project over time could provide many benefits for project stakeholders. For example, stakeholders could monitor the *drift* of a topic, i.e., when the implementation of a topic in the source code gradually diverges from the original design (similar to architectural drift [8]). Because of refactoring, re-engineering, maintenance and other development activities, a topic that was once focused and modularized may become more scattered across the system over time, getting out-of-sync with the mental model that designers and architects have about the system. Automatically discovering and monitoring these topic drifts would be a useful technique for developers and project managers wishing to keep their project in good health.

One could measure the evolution of topics by applying LDA to each version of the system separately and then linking topics together according to a similarity measure (for example, KL divergence [9]). Another method for discovering topic evolution is to apply LDA to all of the versions of the system at once and determine how the values of various topic metrics (e.g., *assignment* or *scattering* [5]) change over time. In this paper, we consider the latter approach, and use the term *topic evolution* to refer to the evolution of the topic's metrics over time.

Although previous work has applied LDA to the history of the source code of a project to recover such topic evolutions [10], it is not yet clear how or why the topics evolve as they do. Topics lie at a higher level of abstraction than other elements found in the source code, such as classes [11], and it has yet to be determined whether the automatically discovered topic evolutions are consistent with the actual changes made by developers (the *change activity*) in the source code. Our goal is to validate the use of topic models to describe software change activities by closely examining the results of the aforementioned previous work.

In this paper, we take a first step towards such a validation by performing a qualitative case study of topic evolution on a well-known and well-documented system, i.e., JHotDraw. We apply LDA to the release history of JHotDraw's source code, compute several metrics on the discovered topics, and manually investigate the source code and project documentation to verify that the evolution of metric values are accurate and consistent with the actual change activity. We use a simple characterization of topic evolutions and find that such topic evolutions are accurate descriptions of the source code evolutions and thus can provide project stakeholders with valuable information for understanding and monitoring their project.

Specifically, the contributions of this paper can be summarized as follows:

1) We find that we can characterize topic evolution by using three simple events: spikes, drops, and stays in metric values over time.
2) We find that most of the discovered topic evolution events arise due to actual change activities in the source code, including bug fixes, refactorings, and feature additions, thus validating the use of topic models for the analysis of software evolution.

This paper is organized as follows. We first provide background information on LDA and topic evolution modeling in Section 2. Section 3 describes our research goals. Section 4 outlines our approach for addressing our research goals, and Section 5 applies our approach to a detailed case study on JHotDraw. Section 6 discusses the our results and enumerates threats to the validity of our study. Section 7 summarizes related work. Finally, Section 8 concludes the paper.

2. Background

2.1. Latent Dirichlet Allocation

LDA is a popular probabilistic topic modeling technique [1]. *Topic modeling* is an automated technique designed to discover *topics* within a corpus of text documents [12], 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—an example topic is *"bank finance money cash loan"*, which describes the financial industry. 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 [12].

As an example, consider the sample documents below:

d_1: "A student left his university to get a loan at the bank."
d_2: "University students prepare for their exams."
d_3: "Banks make money by giving loans."

Applying LDA with $T = 2$ would yield topics similar to:

z_1: "*student university exam*"
z_2: "*bank money loan*"

Document d_1 would have a 50% membership in both topics, since it contains words from both topics to an equal degree, and documents d_2 and d_3 would have a 100% membership in z_1 and z_2, respectively. We could then represent each document as a vector of their topic memberships:

$$d_1 = [0.5, 0.5]$$
$$d_2 = [1.0, 0.0]$$
$$d_3 = [0.0, 1.0]$$

where the first element in each vector corresponds to topic z_1 and the second element to z_2.

More formally, LDA infers, 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 infers a T-dimensional topic membership vector $d(\theta_{1:T})$ that describes the extent to which each topic appears in d. LDA performs these inferences using Bayesian techniques such as collapsed Gibbs sampling [13].

2.2. Topic Evolution in Source Code

Several techniques have been proposed to model collections of text that change over time. Most work to date has focused on traditional corpora such as newspaper articles or conference proceedings, whose topics tend to show only gradual changes over time. Source code topics, on the other hand, may have much larger variability due to the addition of new features or libraries or the removal of whole modules.

The Dynamic Topic Model [14], built for traditional corpora, models the evolution of a topic as a discrete Markov process with normally distributed changes between time periods, which allows only gradual changes

over time. Such a model is thus inappropriate for source code analysis, where the changes to individual topics between time periods could be more dramatic than a normal distribution could allow.

Another proposed model, Topics Over Time (TOT) [15], models time as a continuous beta distribution, allowing for much larger topic changes between successive time periods. 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. [16] 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 time periods. 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.

Linstead et al. have applied the Hall et al. approach to a project's source code release history [10] and found that such a technique can discover topics and their evolution within the source code. It is this work that we build upon in this paper.

3. Research Questions

Topic evolution provides a unique opportunity for automatically monitoring a project's source code over time. For example, if such a monitoring system detected that several topics were becoming more and more scattered across the system, developers may wish to address the issue by refactoring or some other maintenance activity. Automatic monitoring is also useful for answering questions about the history of a project, such as "When was XML functionality first added into the system?" or "At what point did the project switch to a Swing-based UI?"

Previous work has found that topic models, such as LDA, can automatically mine topics from a snapshot of a software system [5], [17] and from its version history [10]. In this paper, we validate these findings with the goal of determining whether the discovered topic evolutions are an accurate description of the actual change activity in the source code.

In particular, we focus on the following research questions:

RQ1 *How can we characterize topic evolution?* What are the characteristics of a topic evolution? How can we describe a topic evolution?

RQ2 *What causes topics to evolve?* What are the underlying driving forces for the evolution of a topic?

Addressing these questions will provide confidence in the use of topic models for studying the evolution of software, bringing us one step closer towards a robust and complete software monitoring technique built upon topic evolution models.

4. Approach

To address our research questions, we investigate the topics and their evolution as produced by the Hall et al. approach. In this section, we formally define our model of topic evolution, explain how we apply LDA to the source code of a software project, and introduce the metrics we use to measure the discovered topics.

Formalization. We say that a ***topic*** z is a tuple $\langle k,\boldsymbol{t},l,\phi\rangle$ discovered by LDA, where k is a unique identifier for the topic, $\boldsymbol{t}$ is the set of top terms related to the topic, l is a one or two word label for the topic, and ϕ is the normalized word distribution over the vocabulary. In this paper, the labels are manually created by the first author, although we note that automated techniques have recently been proposed to label topics [18]. Our manually created labels have the form X::Y, where X is a logically higher-level supertopic, such as UI, and Y is a more specific task within the supertopic, such as MOUSE CLICK.

A ***document*** d in the system is represented by the tuple $\langle n,\boldsymbol{w},\theta\rangle$, where n is the name of the document, $\boldsymbol{w}$ is the preprocessed content of the document (i.e., bag of words—see below for details), and θ is the topic membership vector discovered by LDA. We say that a document d is *related to* a topic z, and that topic z is *present in* document d, if $d(\theta_k) > 0$.

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 in the system, $H = \{V_1, V_2, ...\}$.

Finally, the ***evolution*** E of a metric m of a topic z_k is a time-indexed vector of metric values for that topic:

$$E(m, z_k) = [m(z_k, t(V_1)), ..., m(z_k, t(V_{|H|}))].$$

We also define 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.

Applying LDA. In this paper, we adopt the strategy used by Linstead et al. [10]. In general, we preprocess

the source code, apply LDA to all versions of the code at once to learn a set of topics, and postprocess the results to track individual topics over time.

The preprocessing step involves extracting identifiers and comments (words) from each document. We then tokenize each word based on common naming practices, such as camel case (`oneTwo`) and underscores (`one_two`). We then remove common English language stop words (`the`, `it`, `on`) to reduce noise. Finally, we stem those words that remain to reduce the vocabulary size.

We apply LDA to this entire collection of preprocessed data using the MALLET tool [19], which is an implementation of the Gibbs sampling algorithm. We run the tool for 100,000 sampling iterations, the first half of which are used for optimization [20].

Finally, we postprocess the output of MALLET to group versions of document together and compute various metrics of interest, described next.

Metrics. It is useful to compute several metrics on the output of LDA to measure the topics.

We compute the *assignment* of a topic z_k at version V_i by summing the membership values of that topic over all documents in the version,

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

which gives an indication of the total volume of the topic throughout the code [5]. A higher topic assignment means that more code is related to the topic.

The *scattering* of a topic z_k at version V_i is given by its entropy over all documents [5],

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

Entropy is a common metric used in information theory to determine how uncertain, or spread out, a distribution is [21]. Thus, a topic with a high scatter value will be more spread throughout the system than a topic with a low scatter value.

We introduce a new metric, called the *focus* of a topic z_k at version V_i, which is given by

$$\mathrm{F}(z_k, V_i) = \sum_{j=1}^{|V_i|} \frac{|\{d_{ji}(\theta_k) \geq .5\}|}{|V_i|}.$$

The focus metric reveals how densely, on average, a topic is present in each of the documents that contain it, which cannot be captured by either the assignment or scattering metrics. Topics that are mostly dominant in the documents in which they appear will have a high focus value, while topics that tend to play only a secondary role in the documents will have a low focus value.

5. Case Study

We address our research questions by performing an in-depth case study on a well-known software system, JHotDraw. JHotDraw is a medium-sized, open source, 2-D drawing framework developed in the Java programming language (`http://www.jhotdraw.org`). It was originally developed as an exercise of good program design and has become the de facto standard system for experiments and analysis in topic mining (for example, in [22], [23]). JHotDraw is a good choice for our purposes due to its extensive documentation, good design practices, and manageable size for manual analysis.

In this paper, we considered all 12 public release versions that are available on the JHotDraw project website, giving us a coarse-grained view of the topic evolutions. A revision-by-revision analysis would produce a finer-grained view of the evolution in the system, but in this paper we focus on the coarser-grained view to collect evidence of correctness at this level before going into more detail later on. The 12 versions we considered were released over a nine year period and saw a growth of over 600% in the number of lines of code (from 17K lines in version 5.2.0 to 124K in version 7.4.1), several complete restructurings, and a large number of bug fixes and new feature additions. We modeled JHotDraw with $T = 45$ topics, to be consistent with previous topic modeling analysis on JHotDraw [5].

We applied our approach to the source code of JHotDraw as outlined above, yielding topic evolutions for the three metrics of interest. Table 1 lists a selected subset of topics discovered for JHotDraw.

RQ1: How can we characterize topic evolution?

To address this question, we manually investigate the topic evolutions for each topic over time. We consider various visualizations and later introduce a *change event* classification to characterize the evolutions.

Figure 1 shows a line plot view of the assignment evolutions of a few selected topics that exhibit large fluctuations over time. The evolutions express periods of peaks and valleys, revealing periods of activity and interest in the topic. For example, we see that the TOOL::TEST topic has a large assignment value in versions 5.4.b2 and 6.0.b1, but almost none at any other version. On the other hand, the DRAWING::SHAPES topic slowly grows over time, beginning at version

Topic ID (k)	Topic Label (l)	Word List ($\boldsymbol{t}$)
3	XML::READING	*elem valu attribut read inherit str token ixmlel ioexcept figur color string path stream text*
6	DRAWING::SHAPES	*rectangl width figur pointd grow height draw stroke anchor bound attribut arc connector ellips*
8	UI::SLIDERS	*color map put index compon slider model wheel math string valu set radiu uimanag rgb radial*
25	TOOL::TEST	*test method junit doclet end begin javadoc testcas instanc framework draw set case vault class*
32	TOOL::MENU	*action menu add view applic app window model util jmenu creat item put palett thi bar*
36	UNDO	*undo view command activ undoabl editor set redo execut manag affect select creat wrap dispatch*
37	DISPLAY	*figur draw box displai ifa enumer handl point chang current version framework updat rectangl*

Table 1. Selected topics in JHotDraw. Topic labels are manually assigned; word lists are inferred by LDA.

7.0.7 and continuing until the latest release. Both the DISPLAY and UNDO topics are active during the initial few versions, but disappear at version 7.0.7.

Figure 2 shows a heatmap of the assignment evolutions of all 45 topics. The color of each cell in the heatmap represents the assignment value for a topic at a particular time, where darker colors indicate higher assignment values. This visualization allows us to quickly compare and contrast the trends exhibited by the various topics. For example, the figure indicates that topic 13 (STORAGE::FORMAT) becomes increasingly active during the first four versions of the system, but then dies at the fifth version. On the other hand, topic 17 (UI::TOOLBAR) remains relatively inactive until version 7.2.0, at which point the topic has a large positive change in assignment that remains constant until the latest version. A few topics get increasingly larger assignments (and thus increasingly more code) in the final four releases, as shown by the increasing darkness of the cells (highlighted with thin dashed rectangles and labeled "Increasing" in the figure), while many other topics tend to have near-constant assignments during this time (highlighted with solid rectangles and labeled "Constant" in the figure).

A second use of the heatmap visualization is the ability to visually detect different phases of development. For example, the "visual wall" effect between versions 6.0.b1 and 7.0.7 is created by a large color change in several topics at once (highlighted with thick dotted rectangles and labeled "Spike" in the figure), suggesting that there was a large development effort that affected multiple topics at once. Indeed, version 7.0.7 was a major release with multiple refactorings involving changes to the core framework of the system, as we will investigate further below. A second visual wall is present at version 7.2.0, and the release notes for this version indicate that "substantial changes and enhancements have been made to fix shortcomings and bugs of the frameworks".

Both Figure 1 and 2 suggest that the topics of JHotDraw show signs of active evolution over the releases of the system. These evolutionary changes occur at different times and by different amounts. Some changes are isolated to a few versions, while other changes occur constantly and slowly over time. Some topics do not appear until later in the history, while other topics die.

Change Events. To quantitatively characterize the evolutionary trends of a topic, we introduce the notion of a change event. A *change event* is an increase in metric value (i.e., *spike*), decrease in metric value (i.e., *drop*), or no change in metric value (i.e., *stay*) for a topic between successive versions of a system.

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

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

As a change must be computed by comparing two versions, we do not define a change event for the initial version of a system. Thus, if an evolution contains $|H| = 20$ versions, then there are $|H| - 1 = 19$ change events for each topic. Also, should the denominator of a change c be 0 (i.e., the metric value was 0 at the previous time period) and the numerator is nonzero, we set $c = 1.0$. Note that we assume here that all metric values are ≥ 0.

Table 2 lists the distribution of change events in JHotDraw for the three metrics. We see that a topic's assignment is over three times as likely to spike (41%) than to drop (12%), suggesting that topics tend to grow in size over time. We also see that most topics either drop (36%) or stay (49%) in their scatter metric, indicating that the code has a healthy design and that there is little drift in its location. Finally, we see that the focus metric is twice as likely to experience a spike

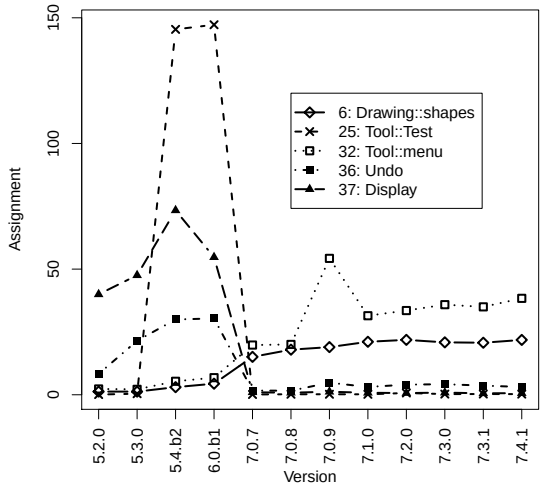


Figure 1. The assignment evolution of five selected topics.

	Spikes	Drops	Stays
Assignment	204 (41%)	58 (12%)	233 (47%)
Scatter	74 (15%)	176 (36%)	245 (49%)
Focus	146 (29%)	68 (14%)	281 (57%)

Table 2. Change events in JHotDraw.

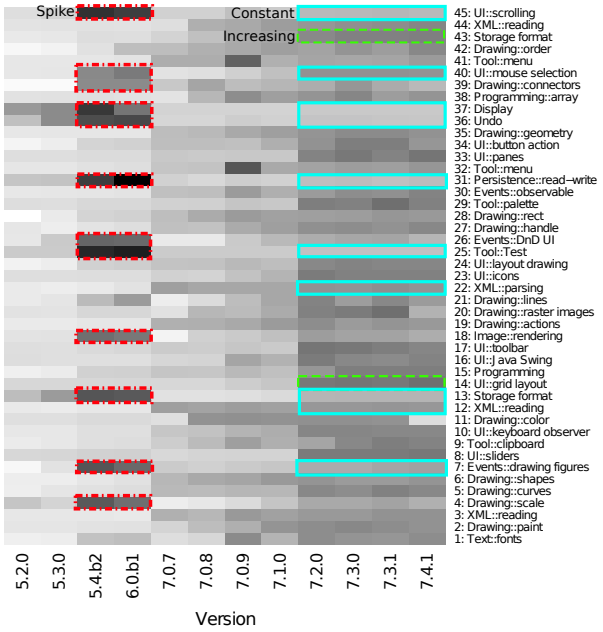


Figure 2. A heatmap view of all topics, showing topic assignment for each version. Darker boxes indicate a higher topic assignment.

(29%) than a drop (14%), suggesting that topics tend to become more dominant and documents tend to become more focused on a single task.

We claim that such change events are simple to compute and understand, yet are able to effectively describe the behavior of a topic's evolution. As such, the change events of a topic evolution are a useful tool for describing software evolution.

We conclude that topic evolutions can be characterized through spikes, drops, and stays in their metric values.

RQ2: What causes topics to evolve?

Longo et al. have recently proposed three categories of software evolution interventions (i.e., causes of software evolution) [24]:

C1) Corrective evolution (i.e., bug fixes)
C2) Refactoring (i.e., code improvement and adaptation)
C3) New functionalities and features

Refactoring is further divided into three categories:

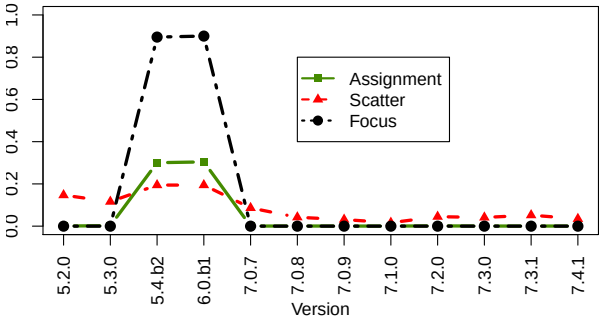
C2.1) Adoption of coding conventions and style
C2.2) Adoption of new framework or libraries
C2.3) Improvement of the internal structure of the code

We refer to each item above as a *change category*. We adopt this taxonomy to investigate why software is changed by developers. Our goal is thus to determine if topic change events correspond to real-world change activities.

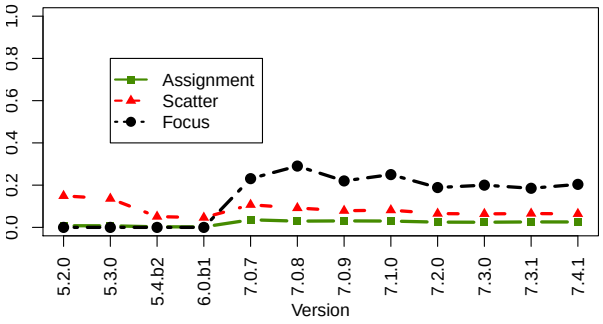
We now qualitatively examine four randomly-selected topic evolutions from JHotDraw in an effort to understand the causes of their evolution. For each selected topic we identify its change events and manually determine their causes by consulting existing documentation. We investigate the release notes, change logs, and source code related the version of the change event, with the goal of determining the underlying cause of the real-world change activity.

The TOOL::TEST Topic. Our first selected topic is one that deals with various testing mechanics used in JHotDraw.

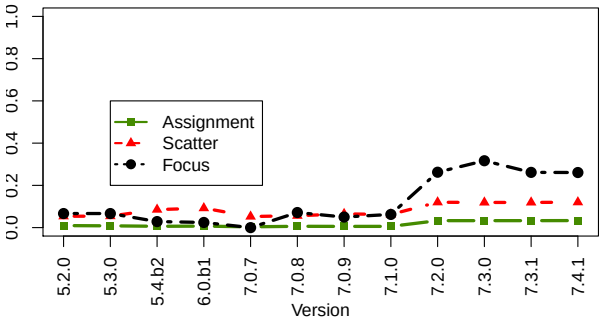
The evolutions of all three metrics for this topic are shown in Figure 3(a). The figure shows a lot of activity in all three metrics between versions 5.4.b2 and 7.0.7: all three metrics show a strong spike, followed by a brief stay, followed by a strong drop. The release notes and source code of these two versions reveal the reasons for this activity. In version 5.4.b2, the developers adopted the JUnitDoclet (`http://www.junitdoclet.org`) testing framework, which automatically creates a skeleton test class for each regular class in an application. This process resulted in a large number of new documents that were specific for the TOOL::TEST topic, resulting in a spike



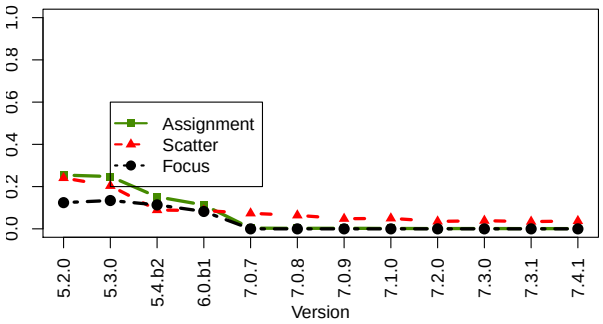
(a) A detailed look at the TOOL::TEST topic.



(b) A detailed look at the XML::READING topic.



(c) A detailed look at the UI::SLIDERS topic.



(d) A detailed look at the DISPLAY topic.

Figure 3. Metric evolutions for the three investigated topics.

in the assignment, scatter, and focus metrics for that topic.

Then, in version 7.0.7, the developers moved away from the JUnitDoclet framework in favor of a more automated technique on top of Ant's JUnit Task framework (`http://www.ant.apache.org`) that is fully automated at build time with no need for saving the generated test classes. When the developers removed the hundreds of test classes created by JUnitDoclet from the release, the assignment, scatter, and focus metrics of the TOOL::TEST topic sharply dropped and remained low for the rest of the system history.

Lesson: A change in frameworks and libraries (change category C2.2) resulted in sharp spikes and drops in all three topic metrics.

The XML::READING Topic. Our next selected topic is one that deals with reading documents in the XML format.

Figure 3(b) shows the evolutions of the topic metrics. We notice a dramatic spike in focus at version 7.0.7, as well as a spike in assignment from almost zero to about 5%, indicating the addition of new code related to the topic. Indeed, the release notes confirm that during this time period, JHotDraw introduced persistency based on XML by implementing a new `org.jhotdraw.xml` wrapper package.

Prior to version 7.0.7, no XML functionality was implemented. However, we see that the scatter of this topic is non-zero during the initial four releases. This is because some documents had non-zero topic memberships for this topic during those releases, an effect of the probabilistic nature of LDA that occasionally matches words in a topic to words in unrelated documents.

An interesting side note for this topic is that between releases 6.0.b1 and 7.0.7, JHotDraw actually had even more XML activity in its inter-release revisions than is shown in Figure 3(b). Namely, at revision 7.0.1, XML functionality was implemented with the help of a third-party library named NanoXML, which was later removed for performance reasons, and XML functionality was re-implemented. In revision 7.0.5, a tweaked version of NanoXML was added back into the system, only to be removed again for release version 7.0.7! This series of events illustrates that a finer granularity of analysis might yield even more interesting results, reinforcing the need to monitor topic evolution revision-by-revision.

Lesson: The addition of functionality (change category C3) resulted in a spike in focus and assignment.

The UI::SLIDERS topic. Our next selected topic is one that deals with slider controls in the user interface.

Figure 3(c) shows the selected topic. Prior to version 7.2.0, there was no standard technique for UI-related classes to implement a slider control. Each class brewed its own flavor of the same slider functionality, an obvious inconvenience to developers and an indicator of a problematic design. In total, we found that five different packages contained such code, none of which included files that were focused on the UI::SLIDERS topic. It is also interesting to note the higher scattering values during this time.

Then, upon the release of version 7.2.0, the `JAttributeSlider` class was introduced that encapsulates slider control functionality into a single class, and the use of sliders increased throughout the GUI. Now, a total of 30 classes across the system contained the UI::SLIDERS topic, all via references to the new class. Eight of the 30 classes were focused, while the remaining 22 were just clients of the `JAttributeSlider` class.

Lesson: Restructuring (change category C2.3) resulted in spikes to the focus metric.

The DISPLAY topic. Our final selected topic is one that deals with displaying a figure on the screen within the drawing box of JHotDraw.

Figure 3(d) shows the selected topic. Initially, this topic contained various figure drawing functionalities scattered across six packages and 255 documents, 26 (10%) of which were focused. In each of the first four releases, many smaller efforts were made by the developers to encapsulate these functionalities into more specific modules, which is shown by the slow decrease in scatter. Then, after release 6.0.b1, a large refactoring occurred that caused almost all of the 255 documents to be removed from JHotDraw, resulting in a sharp drop in all three metrics.

At the same time, four different topics (5, 6, 19, and 35—all in the DRAWING topic category) increased in size by a total of 95%, effectively absorbing the functionality of the now-dead DISPLAY topic. Before the release of 7.0.7, only 97 documents were related to these topics, and none of the documents were focused on any of the topics. After the release, these topics now were present in 190 documents, 32% of which were focused on one of these topics.

We note that the scatter metric is non-zero at times when assignment is almost zero. This is an effect of LDA matching some words from this topic to some words in unrelated documents, a consequence of the probabilistic nature of LDA.

Lesson: Modularization (change category C2.3) caused gradual decreases in the assignment, scattering, and focus metrics.

Manual Validation. In addition to the four example topics presented above, we wanted to manually analyze the rest of the change events in the system. However, since there were a total of 472 change events for the three metrics of the 45 topics, we relied instead on random sampling to select a subset of change events. We thus randomly selected 80 spike and drop change events for manual analysis, corresponding to a 95% confidence level with a margin of error of 5% [25].

We found that almost all (92$\pm$5%) of the randomly selected events corresponded to actual change activities in the source code, while the remaining (8$\pm$5%) had no such correspondence, and were likely caused by noise in the probabilistic LDA model. Of the selected events that corresponded to actual change events, we found that most spikes were due to the addition of functionality (52%) or new frameworks, libraries, or environments (25%). On the other hand, the majority of drops were caused by internal improvements (70%), followed by new coding conventions (11%).

We conclude that topics evolve due to actual change activities in the source code, validating the use of topic models to describe software evolution.

6. Discussion and Threats to Validity

We have found a simple way to characterize topic evolutions, and have found that such characterizations are accurate for the purposes of describing software evolution. These results encourage us to use topic models during software evolution analysis, since topic models are automated, abstract, and accurate.

However, we stress that this paper provides only an initial qualitative assessment of the use of topic models, and as such, our confidence in topic models can be strengthened by additional work.

First, it would be useful to continue the manual analysis on the causes of topic evolutions, eventually analyzing every change event in the system. We posit that current anomalies found in RQ2 are due to noise in the LDA model, but more work is needed to locate additional anomalies and fully understand this effect.

In this work, we investigated whether changes in topic metrics correspond to changes in source code. This work would be further improved by investigating the *other direction*, that is, whether changes in source code cause corresponding changes in topic metrics. Such analysis would add strength to our claims and show an even stronger relationship between source code and topic metrics. We expect such a relationship to hold, as the topics are computed from the source code itself.

As with all source code analysis techniques based on identifiers and comments, our results are dependent on the quality of the naming conventions and commenting style of the project developers. Poor discipline or unorthodox conventions could result in topics with low semantic value. However, a recent study showed that most systems have identifiers and comments that are sufficient for such topic analyses [26].

We have focused our initial analysis efforts on JHotDraw, the de facto standard benchmark for source code topic mining, due to its robust design and fairly complete documentation. However, this could mean that our results are dependent on these qualities and thus generalize poorly to systems with worse designs. Furthermore, as JHotDraw is a medium-sized open source system, we cannot be sure if our results generalize to small or large sized open-source systems, or to any closed-source systems. We also cannot generalize our results with any confidence to systems from different domains. Additional case studies are needed to investigate these alternatives.

7. Related Work

There has been a recent increase in the amount of work involving the use of topic modeling to mine and understand topics within source code. Most of this work aims at making a snapshot of the system easier for the developer to query or understand (e.g., [2], [4], [5]) or to cluster a large collection of systems into functionally related groups (e.g., [27], [28]).

Concern Mining. We note that useful surveys have been created on concern mining techniques that are not based on topic modeling [29], and here we only focus on mining techniques based on topic modeling.

Kuhn et al. introduced *semantic clustering*, a technique based on Latent Semantic Indexing (LSI) to group source code documents that share a similar vocabulary [4]. After applying LSI to the source code, the documents are clustered based on their similarity into semantic clusters, resulting in clusters of documents that implement similar functionalities.

Baldi et al. apply LDA to source code to automatically mine concerns and summarize functionality [5]. They also outlined techniques to compute measurements of scattering and tangling based on the results of LDA.

Linstead et al. used LDA to analyze software evolution, claiming that LDA provides better results than LSI [10]. They showed topic assignment percentages over the version history of a system, which revealed integration points and other changes that shape a project's lifetime. We build on their work by formalizing the approach, considering additional topic metrics (i.e., scatter and focus) to better understand topic change events, and providing a detailed, manual analysis of the topic change events to validate the results of the approach.

Software Clustering. Kawaguchi et al. introduced a tool named MUDABlue that uses LSI to automatically classify software systems into categories based on the identifiers in the system [27]. This is primarily useful for browsing large, unlabeled collections of software.

Tian et al. modified the MUDABlue approach to employ LDA instead of LSI and to consider comments as well as identifiers [28]. They showed that such an approach achieves comparable performance.

8. Conclusion

In this paper we have performed an initial assessment of the validity of using topic models for analyzing software evolution. By applying a topic evolution model to the public releases of JHotDraw and computing three topic metrics of interest, we have shown that the evolution of topics is observable and quantifiable through the changes in their metric values over time. We have further found that most of these metric changes result from actual software change activities, including corrective evolution, internal improvements, and the addition of new features, and thus topic models can be an effective tool for automatically discovering and summarizing such software change activities.

Our case study on JHotDraw suggests that using topic models to study the evolution of the source code of a software project could be useful for developers and other project stakeholders, providing a deep understanding of their system's history and allowing them to monitor and uncover design issues as they appear. Our findings encourage us to further continue this work by exploring revision-level monitoring of topics, defining and measuring additional metrics, and performing additional case studies.

References

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

[2] J. Maletic and A. Marcus, "Supporting program comprehension using semantic and structural information," in *Proc. of the 23rd Intl. Conf. on Software Engineering*. IEEE Computer Society, 2001, p. 112.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] D. E. Perry and A. L. Wolf, "Foundations for the study of software architecture," *SIGSOFT Softw. Eng. Notes*, vol. 17, no. 4, pp. 40–52, 1992.

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

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

[11] M. Lanza, "The evolution matrix: Recovering software evolution using software visualization techniques," in *Proc. of the 4th Intl. Workshop on Principles of Software Evolution*. ACM, 2001, pp. 37–42.

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

[13] I. Porteous, D. Newman, A. Ihler, A. Asuncion, P. Smyth, and M. Welling, "Fast collapsed gibbs sampling for latent dirichlet allocation," in *Proc. of the 14th ACM Intl. Conf. on Knowledge Discovery and Data mining*. ACM, 2008, pp. 569–577.

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

[15] 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.

[16] 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*. ACL, 2008, pp. 363–371.

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

[18] Q. Mei, X. Shen, and C. Zhai, "Automatic labeling of multinomial topic models," in *Proc. of the 13th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*. ACM, 2007, p. 499.

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

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

[21] C. Shannon, "A mathematical theory of communication," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 5, no. 1, pp. 3–55, 2001.

[22] M. Robillard and G. Murphy, "Representing concerns in source code," *ACM Trans. on Software Engineering and Methodology*, vol. 16, no. 1, p. 3, 2007.

[23] D. Binkley, M. Ceccato, M. Harman, F. Ricca, and P. Tonella, "Tool-supported refactoring of existing object-oriented code into aspects," *IEEE Trans. on software engineering*, vol. 32, no. 9, pp. 698–717, 2006.

[24] F. Longo, R. Tiella, P. Tonella, and A. Villafiorita, "Measuring the Impact of Different Categories of Software Evolution," *Software Process and Product Measurement*, pp. 344–351.

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

[26] S. Haiduc and A. Marcus, "On the use of domain terms in source code," in *Proc. of 16th IEEE Intl. Conf. on Program Comprehension*, 2008, pp. 113–122.

[27] S. Kawaguchi, P. Garg, M. Matsushita, and K. Inoue, "Mudablue: An automatic categorization system for open source repositories," *The Journal of Systems & Software*, vol. 79, no. 7, pp. 939–953, 2006.

[28] K. Tian, M. Revelle, and D. Poshyvanyk, "Using latent Dirichlet allocation for automatic categorization of software," *Mining Software Repositories*, pp. 163–166, 2009.

[29] A. Kellens, K. Mens, and P. Tonella, "A survey of automated code-level aspect mining techniques," *Transactions on Aspect-Oriented Software Development*, vol. IV, no. LNCS 4640, pp. 143–162, 2007.