Empir Software Eng (2017) 22:1965–2000
DOI 10.1007/s10664-016-9473-1

Semantic topic models for source code analysis

Anas Mahmoud[1] · **Gary Bradshaw**[2]

Published online: 22 November 2016

Abstract Topic modeling techniques have been recently applied to analyze and model source code. Such techniques exploit the textual content of source code to provide automated support for several basic software engineering activities. Despite these advances, applications of topic modeling in software engineering are frequently suboptimal. This can be attributed to the fact that current state-of-the-art topic modeling techniques tend to be data intensive. However, the textual content of source code, embedded in its identifiers, comments, and string literals, tends to be sparse in nature. This prevents classical topic modeling techniques, typically used to model natural language texts, to generate proper models when applied to source code. Furthermore, the operational complexity and multi-parameter calibration often associated with conventional topic modeling techniques raise important concerns about their feasibility as data analysis models in software engineering. Motivated by these observations, in this paper we propose a novel approach for topic modeling designed for source code. The proposed approach exploits the basic assumptions of the cluster hypothesis and information theory to discover semantically coherent topics in software systems. Ten software systems from different application domains are used to empirically calibrate and configure the proposed approach. The usefulness of generated topics is empirically validated using human judgment. Furthermore, a case study that demonstrates thet operation of the proposed approach in analyzing code evolution is reported. The results show that our approach produces stable, more interpretable, and more expressive topics

Communicated by: Denys Poshyvanyk

✉ Anas Mahmoud
mahmoud@csc.lsu.edu

Gary Bradshaw
glb2@psychology.msstate.edu

[1] Division of Computer Science and Engineering, Louisiana State University, Baton Rouge, LA, 70803, USA

[2] Department of Psychology, Mississippi State University, Mississippi State, MS, 39762, USA

than classical topic modeling techniques without the necessity for extensive parameter calibration.

Keywords Clustering · Information theory · Topic modeling

1 Introduction

Topic modeling is a recent text analysis paradigm that is concerned with exploring, representing, and summarizing electronic text archives (Blei et al. 2003). Topic modeling techniques are used to reduce the dimensionality of a large text corpus down into a set of meaningful topics. Each topic consists of a group of words, collectively representing a *potential* thematic concept that pervades the collection (Hofmann 1999; Blei et al. 2003). Such topics are utilized to support text processing and modeling tasks, providing unique insights into otherwise unseen semantic dimensions in document collections (Hofmann 1999). The unique analytical power of topic modeling has motivated researchers to use such techniques as data analysis models in a plethora of software engineering activities, including: code evolution analysis (Linstead et al. 2008; Thomas et al. 2010), mining software repositories (Linstead et al. 2007), program refactoring (Bavota et al. 2014), statistical debugging (Andrzejewski et al. 2007), bug localization (Lukins et al. 2010), and automated traceability (Asuncion et al. 2010; Mahmoud and Niu 2015). An underlying tenet is that a software system can be analogous to a text corpus, where the textual content of source code, embedded in identifiers, internal comments, and string literals, contains important domain knowledge that captures developers' understanding of the system at the most primitive level (Deißenböck and Pizka 2005; Anquetil and Lethbridge 1998; Haiduc and Marcus 2008). This knowledge can be exploited by semantically enabled topic modeling techniques to discover latent topic structures in the system.

Despite these advances, topic modeling applications in software engineering are frequently suboptimal (Mahmoud and Niu 2015; De Lucia et al. 2012). Problems arise because source code employs a restricted and sparse vocabulary that often lacks uniqueness and exhibits a high degree of repetition (Hindle et al. 2012). This leads data intensive topic modeling techniques, such as Latent Dirichlet Allocation (LDA) (Blei et al. 2003) and Latent Semantic Analysis (LSA) (Kuhn et al. 2007), to perform poorly when applied to source code (De Lucia et al. 2012). Another limitation stems from the underlying mathematical complexity of such techniques. In particular, several parameters have to be tuned simultaneously via a computationally expensive process in order to identify near optimal configurations that work best for a specific system, or even a specific task (Andrzejewski et al. 2007; Panichella et al. 2013; Lohar et al. 2013; Mahmoud and Niu 2015). This complexity raises major concerns about the feasibility of using topic modeling techniques to support automated software engineering solutions (Mahmoud and Niu 2015).

Motivated by these observations, in this paper, we introduce a new topic modeling approach designed for software systems. The proposed approach, which we refer to as Semantic Topic Models (STM), draws upon the basic assumptions of the cluster hypothesis (van Rijsbergen 1979) and information theory (Cilibrasi and Vitanyi 2007). In the context of text processing, the cluster hypothesis suggests that words with similar meanings tend to be grouped in the same cluster. The notion of similarity between words can be estimated based on their mutual information (Church and Hanks 1990; Cilibrasi and Vitanyi 2007). More specifically, STM exploits the co-occurrence cues of individual terms, extracted from source

code, to estimate their pairwise semantic similarity. These terms are then clustered to produce multinomial distributions of words, representing potential domain topics and system concerns. Individual terms of each topic are assigned weights derived from their potential information value to the topic. Each system's artifact is then assigned to the set of generated topics based on its textual semantic similarity to each topic. Ten open source software systems from different application domains are used to configure the proposed approach. Human judgment is then used to assess the quality of the generated topics in comparison to topics generated using classical topic modeling techniques. Furthermore, a case study that demonstrates the operation of the proposed approach in the domain of code evolution analysis is reported. In particular, in this paper, we document two main contributions:

– A stable and easy to operate topic modeling technique designed for source code.
– An adaptation of popular human driven topic modeling evaluation strategies to assess the performance of topic modeling techniques in the context of source code analysis.

The remainder of this paper is organized as follows. Section 2 briefly introduces topic modeling, its main applications and limitations in software engineering, and motivates our research. Section 3 describes the proposed approach. Section 4 presents our experimental evaluation and analysis results. Section 5 discusses our main findings. Section 6 describes a case study that applies the proposed approach to code evolution analysis tasks. Section 7 discusses the main threats to the study's validity. Section 8 reviews related work. Finally, Section 9 concludes the paper and discusses prospects for future work.

2 Background and Motivation

This section describes several topic modeling techniques that are often used in source code analysis, discusses the main limitations of topic modeling in software engineering applications, and motivates our research.

2.1 Topic Modeling

A wide variety of topic modeling techniques have been proposed in the Natural Language Processing (NLP) literature (Hofmann 1999; Deerwester et al. 1990; Blei et al. 2003). Origins of topic modeling can be traced back to Latent Semantic Analysis (LSA). LSA is an unsupervised statistical algorithm that is used for extracting and representing relations between words and documents in large unlabeled text corpora (Deerwester et al. 1990). LSA is based on the assumption that there is some underlying (*latent*) semantic structure that is partially concealed by the variability of the contextual usage of words in a text collection (corpus) (Deerwester et al. 1990). Formally, LSA starts by constructing a $m \times n$ $word \times document$ matrix (χ) for words and documents in the corpus. This matrix is usually huge and sparse. Rows represent unique words in the corpus and columns represent textual artifacts (full documents or pieces of text). A specific weight is assigned to each term in the matrix. Such weights can range from simple word counts in documents, to more semantically aware schemes, such as TF.IDF weights of words (Salton et al. 1975). Singular Value Decomposition (SVD) is applied to decompose the $word \times document$ matrix χ into a product of three matrices $\chi = USV^T$ (Demmel and Kahan 1990). $\boldsymbol{S}$ represents a diagonal matrix of singular values arranged in a descending order, and both $\boldsymbol{U}$ and $\boldsymbol{V}$ are column orthogonal matrices. Dimensionality reduction is then performed to produce reduced approximations

of USV^T by keeping the top k eigenvalues of these matrices. More specifically, the dimensions of $\boldsymbol{S}$ are reduced by removing singular values that are too small, only keeping the initial principal components. Reducing the dimensionality of the observed data will smooth out the data, which results in relatively better approximations to human cognitive relations (Deerwester et al. 1990). The reduced matrix can be described as $\chi_k = U_k S_k V_k^T$, where χ_k represents a compressed matrix of rank k that minimizes the sum of the squares of the approximation errors of χ. Since $U_k S_k V_k^T$ are $m \times k$, $k \times k$, and $k \times n$, respectively, χ_k is still a $m \times n$ matrix similar to χ. Using χ_k, the similarity of two words in the corpus $< w_1, w_2 >$ can be measured as the cosine of the angle between their corresponding compressed row vectors. In the context of information retrieval (IR), LSA is often known as LSI (Latent Semantic Indexing). LSI considers document vectors in the LSA semantic space, where a document is represented as a vector of weighted corpus words. The similarity between two documents can then be calculated as the cosine similarity between their corresponding vectors in the LSI SVD reduced space.

In an attempt to generate more statistically sound results of LSA, Hofmann et al. (1999) proposed probabilistic LSA (PLSI), an unsupervised generative probabilistic approach which models each word in a document as a sample from a mixture model. A topic ϕ_k in PLSI is a multinomial distribution over a fixed size vocabulary V. Thus, a topic is a vector in which each element $\phi(k, w)$ represents the probability that a term w belongs to the topic k. Similarly, each document d is described as a vector (multinomial distribution) θ_d in which each element represents the probability that a topic k contributes to the document's subject matter. The generative process of PLSI can be described as deciding for each word in a document which topic to choose from θ_d and then which word to choose from the selected topic ϕ_k. EM (Expectation Maximization) is employed to estimate these unknown parameters by maximizing the log likelihood of the training data.

In PLSI, the number of parameters in the model grows linearly with the size of the corpus. This often leads to overfitting problems caused by these too many unseen variables (Girolami and Kabán 2003). To overcome these limitations, Blei et al. (2003) introduced Latent Dirichlet Allocation (LDA) (Maskeri et al. 2008; Thomas et al. 2010; Meghan et al. 2009). LDA reduces the dimensionality of a large text corpus down into a set of meaningful latent topics. Each topic consists of a group of words that represents a potential thematic domain concept of the corpus. Formally, LDA is an unsupervised probabilistic approach for estimating a topic distribution over a text corpus. The main assumption is that documents in the corpus are generated using a statistical generative model as random mixtures over latent topics. Therefore, the set of latent topics that are likely to have generated the documents can be inferred from the corpus's text by simply inverting its generative model (Hofmann 1999). To formally explain LDA's underlying mathematical model, we start with the following definitions:

Definition 1 A word w is the basic unit in a vocabulary of size W

Definition 2 A document d is a sequence, or a subset, of words sampled from W, such that $d = \{w_1, w_2, w_3, ..., w_n\}$. A word w_i might appear multiple times in the document d.

Definition 3 A corpus D is a set of documents, such that $D = \{d_1, d_2, d_3, ..., d_m\}$

Assuming K number of topics, each topic t_i in the latent topic space ($t_i \in T$) is modeled as a multidimensional probability distribution, sampled from a Dirichlet distribution β, over the set of unique words (W) in the corpus, such that $\phi_{w|t} \sim Dirichlet(\beta)$, where

β is a Dirichlet prior on the topic word distribution. Similarly, each document d_i from the collection D, is modeled as a probability distribution, sampled from a Dirichlet distribution α over the set of topics, such that $\theta_{t|d} \sim Dirichlet(\alpha)$, where α is a Dirichlet prior on the document topic distribution.

For the i^{th} word in a document d, a topic index z_{id} is drawn from the topic weights such that $z_{id} \sim Multinomial(\theta_{t|d})$ (i.e., z_{id} is the topic index of the n^{th} word in the d^{th} document). In addition, the observed word w_{di} is drawn from the corresponding topic such that $w_{di} \sim Multinomial(\phi_{w|z_{id}})$. This generative model can be described up as follows:

- For k = 1, ..., K
  $\phi_k \sim Dirichlet(\beta)$

- For each $d \in D$
  $\theta_{t|d} \sim Dirichlet(\alpha)$

- For each $w \in d$
  $z_{id} \sim Multinomial(\theta_{t|d})$
  $w_{id} \sim Multinomial(\phi_{w|z_{id}})$

In the above model, words within documents serve as observed data, the known parameters of the model include the number of topics K, and the Dirichlet priors on the word-topic and topic-document distributions β and α. The remaining parameters ($\theta_{t|d}$ and ϕ_k) must be inferred. Inference is used to reverse the generative process and learn the posterior distributions of the latent variables ($\theta_{t|d}$ and ϕ_k) in the model given the observed data (words of documents). Since exact inference for this model is intractable (Blei et al. 2003), approximate inference techniques can be used.

From among the several approximate posterior inference algorithms that have been proposed to learn an LDA model (e.g., Variational Bayesian (VB) (Blei et al. 2003) and Collapsed Variational Bayesian CVB (Teh et al. 2007)), Gibbs Sampling (GS) has gained the most attention. Gibbs sampling is an inference technique that samples conditional distributions of the variables of the posterior (Griffiths and Steyvers 2004; Porteous et al. 2008). The sampling algorithm goes through the words of each document d_i in the collection D (observed data) and randomly assigns words to one of the K topics. This process creates an initial, naturally weak, full assignment of words and documents to topics. To improve these assignments, the sampling process proceeds by iterating through each word in each document, assuming that all topic assignments except for the current word are correct, the assignment of the current word is then updated using the model. This learning process is repeated multiple times until the samples begin to converge to a stationary distribution. Formally, Gibbs sampling can be described as follows:

$$p(z_{id} = t | w_{id} = w, z^{-i}) \propto \frac{N_{wt}^{-i} + \beta}{N_t^{-i} + W\beta}(N_{td}^{-i} + \alpha) \tag{1}$$

where z^{-i} is the set of topic assignment variables except the i^{th} variable (leaving the i^{th} word out of the calculation), N_{wt} is the number of times a word w has been assigned to topic t, N_{td} is the number of times topic t has been assigned to document d, and $N_t = \sum_{w=1}^{W} N_{wt}$. A detailed derivation of Gibbs sampling can be found in Griffiths and Steyvers (2004).

Convergence is theoretically guaranteed with Gibbs Sampling. However, there is no practical way of determining the exact number of iterations required to reach the stationary distribution (i.e., exact inference is not computationally feasible). Therefore, an LDA model is typicality extracted when the sampling reaches an acceptable estimation of convergence (Blei et al. 2003). In particular, the generated assignments are used to estimate the topic mixtures of each document (the proportion of words assigned to each topic within that document) and the words' associations to each topic (the proportion of words assigned to each topic). This process generates for each topic t the matrix $\phi_t = \{\phi_1, \phi_2, \ldots, \phi_{|W|}\}$, representing the distribution of t over the set of words in W, and for each document d the matrix $\theta_d = \{\theta_1, \theta_2, \ldots, \theta_K\}$, representing the distribution of d over the set of learned topics. The model takes the text corpus, the number of topics K, and the hyperparameters α and β, and the *burn in* (number of iterations) as input, and returns the $word \times topic$ and the $topic \times document$ matrices as output.

2.2 Limitations and Motivation

Topic modeling techniques have been applied across a broad range of software engineering applications (see Section 8.1) (Linstead et al. 2008; Thomas et al. 2010; Linstead et al. 2007; Bavota et al. 2014; Andrzejewski et al. 2007; Lukins et al. 2010; Asuncion et al. 2010; Mahmoud and Niu 2015). Despite these advances, important concerns have been raised about several limitations of topic modeling techniques that hinder their adaptation as tools for source code analysis. Such limitations can be described as follows:

- Calibration: Most state-of-the-art topic modeling techniques require an exhaustive calibration of several parameters in order generate the desired output (Blei et al. 2003; Kuhn et al. 2007; Hofmann 1999). Such parameters (e.g., α, β, K, and number of iterations) often need to be tuned simultaneously until the configuration settings that best fit a specific system, or a task, are identified. In most cases, researchers attempt to overcome this problem by relying on experimental heuristics (Andrzejewski et al. 2007; Grant and Cordy 2010; Abadi et al. 2008) or by running the topic modeling algorithm multiple times and then take the average (or best) performance over the multiple runs. However, such heuristics are not necessarily guaranteed to work for all experimental settings. Furthermore, there is no objective way to determine how many times to run the algorithm or how to select the best model. In fact, even with the support of automated calibration strategies (Lohar et al. 2013; Panichella et al. 2013), such a process is still computationally expensive and not guaranteed to find an optimal solution.
- Implementation complexity: The underlying mathematical structure of generative topic modeling techniques are often unintuitive to understand (Blei et al. 2003). This non-trivial theoretical complexity is a main reason that topic modeling is often used as an *of-the-shelf* blackbox with little to no customization to handle special corpora such as software systems (De Lucia et al. 2012; Abadi et al. 2008).
- Stability: Topic modeling techniques, such as LDA and PLSI, rely on intractable stochastic inference strategies to generate topics. Therefore, there is no guarantee that different runs of such techniques will generate similar topic distributions (Wallach et al. 2009; Blei et al. 2003). This raises major practicality concerns as solutions (generated models) might drastically vary depending on the initial inference settings.
- Data sparsity: Source code often suffers from data sparsity. In particular, the lexicon from which source code text is drawn is limited. Therefore, source code often

lacks uniqueness and exhibits a large degree of repetitiveness, even more than natural language (Gabel and Zhendong 2010; Hindle et al. 2012). From an analytical perspective, the lack of sufficient amounts of data raises major concerns about the feasibility of using data intensive statistical methods (e.g. LDA, LSA) to generate meaningful, or semantically coherent, topics in source code (Abadi et al. 2008; De Lucia et al. 2012; Maskeri et al. 2008; Mahmoud and Niu 2015). These observations pose another challenge on employing topic modeling techniques to synthesize source code and emphasize the need for nontraditional measures to capture the notion of semantic coherence in software systems.

Motivated by these limitations, and the growing need for a robust topic modeling approach that can address the inherent characteristics of source code, in this paper, we propose an information theoretic approach for discovering semantic topics in software systems. The proposed approach is stable —different runs generate similar topic distributions. Our approach also requires minimum calibration. More specifically, a self optimizing procedure is used to automatically look for a solution in the topic space (cluster hierarchy) guided by a quality objective function that captures the semantic coherence of generated topics. Furthermore, the proposed approach exhibits moderate space complexity as it works with one main matrix of raw co-occurrence data to create topics.

3 Approach

The proposed approach in this paper draws on the basic assumptions of the cluster hypothesis, concepts of information theory, and recent findings in NLP topic modeling research. Our main objective is to describe a topic modeling approach that can be as analytically insightful as classical topic modeling techniques, and yet, does not suffer from the limitations described earlier. More specifically, the proposed approach, depicted in Fig. 1, can be described as a multi-step procedure. Initially, the textual content of source code is extracted

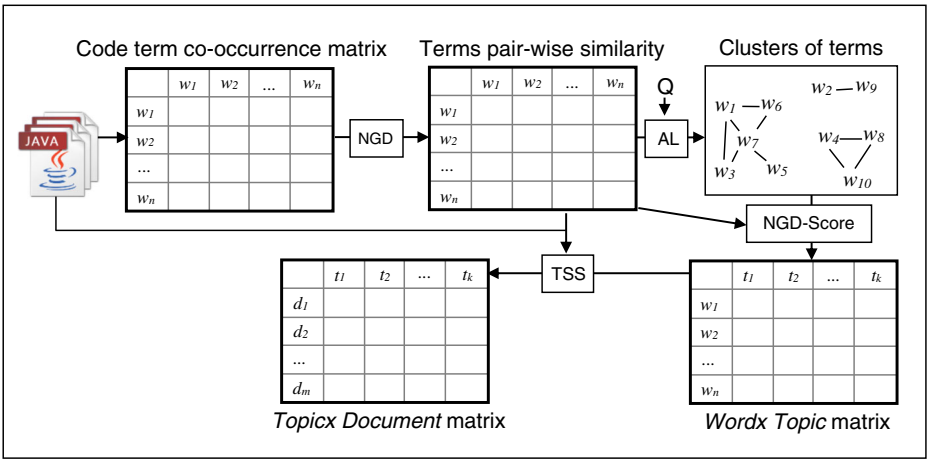

Fig. 1 The proposed semantic topic modeling technique. NGD: Normalized Google Distance, AL: Average Linkage clustering algorithm, Q: Topic quality function, TSS: Text Semantic Similarity)

and tokenized. The co-occurrence matrix of extracted terms is constructed based on their co-occurrence patterns over the system's artifacts. The semantic similarity between individual code terms is estimated based on their mutual information. Terms are then clustered to generate a set of topics. Individual topic terms are assigned weights corresponding to their information value to the topic, and system artifacts are assigned to topics based on their semantic similarity to each topic's subject matter. In what follows, we describe these main steps in greater detail, including the main theoretical assumptions, research questions, and experimental analysis associated with each step.

3.1 Source Code Indexing

The first step in our procedure is to index source code. Indexing is the process of extracting and tokenizing the textual content of code, embedded in identifier names, internal comments, and string literals. In our analysis, a code file is treated as a raw text file. The textual content of each file is extracted using standard string manipulation. Extracted code identifiers are split into their constituent terms based on standard camel casing (e.g., `UserID` is split to `User` and `ID`). This step can be avoided if the analyst is interested in full identifier names rather than their primitive vocabulary. Programming reserved words and English stop-words are then filtered out (e.g., `int`, `the`). The stop-word list provided by StanfordNLP[1] is used in our analysis. The remaining terms are stemmed to their morphological roots using Porter stemmer (Porter 1997). The outcome of the indexing process is compact content descriptors (term vectors) of each code artifact.

In our analysis, we work at a class granularity level (i.e., a code artifact is basically a single class). In particular, since we are dealing mainly with Object Oriented systems, we assume that each file holds a single class. An inner class is considered to be a part of the main class. Figure 2 shows the outcome of the indexing process over a sample code segment. Specific details about the code indexing process can be found in our code indexing tool STAC[2] (Khatiwada et al. 2016).

3.2 Estimating Semantic Similarity between Source Code Terms

The second step in the proposed approach is to calculate the pairwise semantic similarity between individual code terms extracted from the system. This step is crucial to the accuracy of the proposed approach since the pairwise similarity will determine how these terms will be grouped into topics. Our objective is to create semantically coherent topics that encapsulate meaningful and discriminating domain concepts.

Several attempts have been made in NLP research to capture the semantic coherence of linguistic topics. Newman et al. (2010) reported that higher average of term pairwise semantic similarity, quantified based on term co-occurrence patterns in massive corpora such as Wikipedia[3] or Google n-grams,[4] gives a strong indication of higher quality (more useful) topics. Similarly, Mimno et al. (2011) reported that most types of poor quality topics could actually be detected using a metric based on the co-occurrence of words within the corpus being modeled, and Lau et al. (2010) reported that the co-occurrence patterns of a

[1] http://nlp.stanford.edu/

[2] https://github.com/seelprojects/STAC

[3] https://dumps.wikimedia.org/

[4] https://books.google.com/ngrams

```
//Sets up the attributes for the class
   public AddHCPAction(DAOFactory factory, long loggedInMID) {
      this.personnelDAO = factory.getPersonnelDAO();
      this.authDAO = factory.getAuthDAO();
   }
```

```
set up attribut
add hcp action dao factori factori log mid
personnel dao factori get personnel dao
auth dao factori get auth dao
```

Fig. 2 Source Code Indexing

topic's words can be used as a feature for generating representative labels for the topic. In general, the converging evidence indicates that the average semantic similarity of a certain topic's words, captured through the distributional cues of words over the corpus, seems to be the most accurate representative of the semantic coherence of the topic (Newman et al. 2010; Mimno et al. 2011). In Newman et al. (2010) and Lau et al. (2010), Pointwise Mutual Information (PMI) is used to quantify such information. PMI is an information theoretic measure of information overlap, or statistical dependence, between two words (Church and Hanks 1990). Formally, assuming a text collection contains N documents, $C(w_1, w_2)$ is the number of documents in the collection containing both w_1 and w_2, and $C(w_1)$ and $C(w_2)$ are the number of documents containing w_1 and w_2 respectively, PMI between w_1 and w_2 can be measured as:

$$PMI = \log_2\left(\frac{\frac{C(w_1,w_2)}{N}}{\frac{C(w_1)}{N}\frac{C(w_2)}{N}}\right) \tag{2}$$

In this equation, N can be factored out as it is the same for all PMI pairs. Moreover, since we are interested in the relative ranking, (2) can be simplified further as shown in (3) (Budiu et al. 2007; Recchia and Jones 2009):

$$PMI \approx \log_2\left(\frac{C(w_1, w_2)}{C(w_1)C(w_2)}\right) \approx \frac{C(w_1, w_2)}{C(w_1)C(w_2)} \tag{3}$$

Due to the inherent differences between programming languages and natural languages (Hindle et al. 2012), PMI cannot be applied to code without adjustment. For instance, PMI over values sparseness, thus, it often crumbles with smaller corpora (Recchia and Jones 2009). More specifically, in situations where there is a perfect term dependence (two terms only appear together), the rarer the terms are, the higher the PMI is. Thus, a higher PMI score does not necessarily mean higher word dependence. Given the sparsity and lack of uniqueness of source code (Gabel and Zhendong 2010; Hindle et al. 2012), the term dependency problem tends to be more frequent, thus taking PMI's accuracy to lower levels.

To overcome this problem, we consider another co-occurrence based measure of semantic similarity, known as the Normalized Google Distance (NGD) (Cilibrasi and Vitanyi 2007; Mahmoud and Bradshaw 2015). NGD uses Google counts to estimate semantic similarity between natural language words based on information distance and Kolmogorov

complexity (Li et al. 2004). Formally, the semantic distance between two words w_1 and w_2 can be calculated as:

$$NGD(w_1, w_2) = \frac{\max\{\log(D_1), \log(D_2)\} - \log(|D_1 \cap D_2|)}{\log(|D|) - \min\{\log(D_1), \log(D_2)\}} \quad (4)$$

In this equation, D_1 and D_2 are the number of documents in the corpus containing w_1 and w_2 respectively, $|D_1 \cap D_2|$ is the number of documents containing both w_1 and w_2, and D is the total number of documents in the corpus. NGD tends to be immune to the perfect dependency problem. Mathematically, all terms with perfect dependency (i.e., only appear together) generate a Google distance of 0 (similarity of 1), regardless of their frequencies, thus overcoming the problem of data sparsity affecting PMI. In order to bound NGD value in the range [0, 1], where 1 means perfect similarity, the following formula is often used (Gracia et al. 2006):

$$NGD = e^{-2NGD(w_1, w_2)} \quad (5)$$

In our analysis, we adopt (5) as the main measure of semantic similarity between code terms. However, instead of using Google counts to get the co-occurrence information of terms, we rely on the co-occurrence patterns of terms across the system itself. This design decision is based on the observation that external sources of textual knowledge (e.g., Wikipedia and Google n-grams (Newman et al. 2010)) cannot be used to estimate semantic similarity between code terms. More specifically, code terms that are semantically related within the context of a software system may not necessarily be semantically related in English and vice versa (Sridhara et al. 2008; Mahmoud and Bradshaw 2015). In addition, a considerable percentage of code terms are domain specific (e.g., domain coined acronyms) (Caprile and Tonella 2000), thus, do not appear in such external corpora.

3.3 Generating Topics

The third step in the proposed approach is to group code terms into semantically coherent clusters based on their semantic similarity. The cluster hypothesis suggests that documents in the same cluster behave similarly with respect to relevance to information needs (van Rijsbergen 1979; Niu and Mahmoud 2012). This behavior can be tied to the information content of documents embedded in their individual words. In particular, words with similar meaning tend to be grouped in the same cluster (Hearst and Pedersen 1996). Formally, the main task at this step is to cluster a group of code terms $W = \{w_1, w_2, w_3, ..., w_m\}$ into a set of disjoint groups of semantically similar terms $C = \{c_1, c_2, c_3, ..., c_k\}$, where each term w_i can belong to only one cluster c_j. Ideally, clusters in C should describe conceptual themes (topics) that pervade the system.

Several word clustering algorithms have been proposed in the literature (Slonim and Tishby 2000). These algorithms apply various merging schemes to group words into meaningful clusters. In our analysis, we adopt Hierarchical Agglomerative Clustering (HAC) as our main clustering mechanism (Schaeffer 2007). The underlying logic of agglomerative clustering is to successively merge the most similar elements in a data space into larger clusters. HAC clustering algorithms produce hierarchical structures that can be represented as dendrograms (Anquetil et al. 1999). The most popular variants of HAC include Complete Linkage (CL), Single Linkage (SL), and Average Linkage (AL). These algorithms vary in the way they calculate distance between clusters. Formally, assuming the distance between any two data items in two clusters **A** and **B** is given by $d(a, b)$, the linkage (merging) criteria for clusters **A** and **B** can be described as follows:

– SL: $M(A, B) = min\{d(a, b) : a \in A, b \in B\}$

– CL: $M(A, B) = max\{d(a, b) : a \in A, b \in B\}$
– AL: $M(A, B) = average\{d(a, b) : a \in A, b \in B\}$

The main research question at this point is how to determine an optimal number of clusters (topics) for a specific system. In techniques such as LDA or PLSI, the number of topics (K) is often determined manually, based on expert advice or existing heuristics (Blei et al. 2003). In general, K should maximize the semantic coherence and distinctivity of generated topics. Semantic coherence can be described by cohesion, which is a measure of compactness, or how semantically related, words in a cluster are. Distinctivity, on the other hand, can be described by coupling, or how semantically separated clusters are from one another. An optimal number of clusters should maximize coherence and distinctivity.

To quantify such information, we derive a topic quality measure that is based on the observation that topics with higher average semantic similarity between their words tend to be more semantically coherent (i.e., more meaningful) (Newman et al. 2010; Mimno et al. 2011; Lau et al. 2010). Formally, in the proposed approach, the semantic coherence of a cluster c_i with n terms is calculated as the average pairwise semantic similarity between its code terms:

$$cohesion(c_i) = \frac{1}{\binom{n}{2}} \sum_{w_i, w_j \in c} NGD(w_i, w_j) \tag{6}$$

The separation, or coupling, of a cluster is quantified as the average pairwise term semantic similarity (NGD) between a cluster and its nearest neighbour in the generated cluster space. Formally, assuming M number of clusters, coupling of c_i is calculated as follows:

$$coupling(c_i) = \max_{0 \leq j \leq M,\ j \neq i} sim(c_i, c_j), \tag{7}$$

where $sim(c_i, c_j)$ is the average term pairwise NGD similarity between clusters. The overall quality of the generated decomposition is then calculated as:

$$Quality(C) = \frac{1}{|C|} \sum_{i=1}^{|C|} (cohision(c_i) - coupling(c_i)) \tag{8}$$

The value of this semantically aware quality metric fits in the range [-1, 1]. 1 indicates a theoretically perfect scenario, where cohesion is maximized and coupling is minimized, and -1 is a worst case scenario, where formed clusters have no semantic coherence.

We select ten open source systems from different application domains to evaluate of the different clustering algorithms using the objective quality function in (8). Our main objective is to empirically determine whether any of the HAC algorithms can actually generate a solution (converge), given NGD (5) as a similarity measure. Our list of experimental systems include: iTrust,[5] Ivy,[6] JHotDraw,[7] JavaHMO,[8] iText,[9] MegaMek,[10] Prefuse,[11] JEdit,[12]

[5] http://agile.csc.ncsu.edu/iTrust/wiki/doku.php

[6] http://ant.apache.org/ivy/

[7] https://sourceforge.net/projects/jhotdraw/

[8] http://javahmo.sourceforge.net/

[9] http://developers.itextpdf.com/itext-java

[10] https://github.com/MegaMek

[11] http://prefuse.org/download/

[12] http://www.jedit.org/

Table 1 Experimental Systems

SYSTEM	KLOC	COMMENTS	CLASS	TERMS	VER.
iTrust	47.6	12.7	687	3,020	15.0
Ivy	39.9	14.3	386	2,856	2.3.0
JHotDraw	32.1	18.7	309	2,718	7.0.6
JavaHMO	25.6	7.8	166	2,865	2.4
iText	94.1	70.4	549	9,820	5.0
MegaMek	87.9	280.6	1,863	10,587	0.40.1
Prefuse	40.9	27.5	403	4,448	2007.10.21
JEdit	118.5	48.4	573	6,191	5.2
Ant	135.7	102.7	1,216	8,583	1.9.6
JBidwatcher	28	6.7	231	4,106	2.99pre5

Ant,[13] and JBidwatcher.[14] Table 1 describes these systems, including the number of kilo lines of code (KLOC), kilo lines of comments (COMMENTS), the number of classes (CLASS), number of unique terms (TERMS) extracted from each system, and the specific version VER. used in our analysis.

To get a sense of NGD's performance as a distance measure, we compare its performance to LSA (i.e., LSA is used to build the term pairwise similarity matrix of the system and as the main similarity measure in (6) and (7)). A $document \times term$ matrix, calculated using the TF.IDF weights of terms, and a term co-occurrence matrix are constructed for each system. These two matrices are then used to generate the LSA and NGD term pairwise similarity matrices for each system. A brute force optimization strategy is then applied to identify near optimal combinations of the different clustering algorithms and distance measures introduced earlier. In particular, the LSA and NGD term pairwise similarity matrices are fed to the different clustering algorithms. At each level of the generated cluster hierarchy, a cut is taken and the quality (8) is measured. The cut point in the dendrogram that maximizes the value of our quality function is considered the point of convergence. Ideally, an effective clustering algorithm should converge to a global maximum at a certain cut point. This point represents the optimal number of topics for the system. Note that optimizing the performance of the different clustering algorithms, using LSA as the distance measure between words, requires also optimizing the K value for the space reduction step of LSA. To achieve this, K is initially set to $K = 10$, the LSA term pairwise similarity matrix is then generated and the above optimization process is repeated. The value of K is gradually increased by 10 and the process is repeated until K is equal to the number of singular values in the matrix $\boldsymbol{S}$ (See Section 2.1).

The results of the optimization process over all of our experimental systems are shown in Table 2 and Fig. 3. Table 2 shows the maximum quality achieved by each clustering algorithm over each system, the number of clusters (K topics) at which this value was achieved, and the $LSA - K$ value at which LSA achieved its best results. In general, the results show that AL was the most successful in extracting highly cohesive clusters in all systems.

[13] http://ant.apache.org/

[14] https://www.jbidwatcher.com/

Table 2 Best performance in terms of quality (8) of the different clustering algorithms using NGD and LSA

	NGD				LSA				
SYSTEM	CL	SL	AL	K	CL	SL	AL	K	LSA-K
iTrust	0.21	0.02	0.34	103	0.09	0.01	0.22	97	250
Ivy	0.38	0.01	0.40	144	0.09	0.01	0.24	111	110
JHotDraw	0.04	0.04	0.34	62	.01	0.01	0.21	67	170
JEdit	0.34	0.00	0.40	100	0.22	−0.04	0.14	113	100
iText	0.4	0.01	0.44	264	0.21	−0.04	0.16	210	230
MegaMek	0.24	0.02	0.50	217	0.28	−0.02	0.17	137	200
JavaHMO	0.13	0.08	0.48	53	0.36	0.00	0.40	66	50
Ant	0.21	0.02	0.42	194	0.11	−0.07	0.32	203	70
Prefuse	0.28	0.05	0.47	104	0.21	−0.08	0.21	114	120
JBidwatcher	0.21	0.04	0.52	113	0.31	0.00	0.22	98	140

AL reached an average quality level of 0.46, in compassion to 0.22 and 0.11 for SL and CL respectively, confirming previous observations about the suitability of AL for text clustering (Aggarwal and Zhai 2012). In general, AL tends to produce relatively balanced term clusters, where a term has to have a high average pairwise similarity to all other terms in the cluster in order to be merged in, thus preserving the context, or the theme, of the topic (Aggarwal and Zhai 2012). On the other hand, CL and SL rely on a single term pair to determine the distance between two clusters. This often affects the term clustering process negatively since other terms that might be more distinctive to the topic may be ignored. The results also show that CL consistently outperforms SL in all systems. This can be explained based on the underlying merge rule of each algorithm. In particular, SL calculates the similarity between two clusters as the similarity of their most similar members, causing bigger clusters to be grouped together rather than incorporating singletons. Therefore, SL tends to create a small number of large, isolated clusters. Due to their size, such clusters tend to exhibit very low cohesion. In contrast, CL tends to push clusters apart, creating a large number of small, but highly cohesive, clusters. The results also show that LSA, while being able

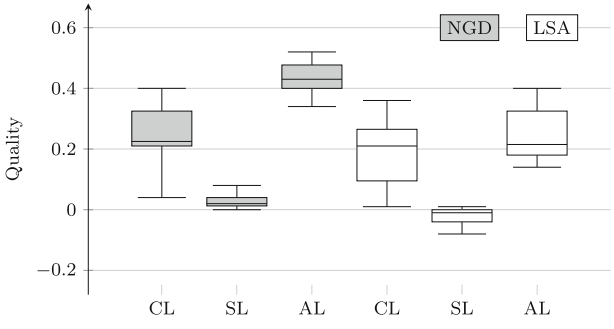

Fig. 3 Aggregated performance of the different clustering algorithms (CL, SL, and AL) in terms of quality (8) using NGD (gray boxes) and LSA (white boxes)

to compete with NGD in some systems, was not able to match the average performance of NGD. LSA achieved an overall quality of 0.13 (average performance of all clustering algorithms across all systems), in comparison to NGD's average quality of 0.23. These results provide further evidence of the limitations of applying data intensive techniques, such as SVD, to model sparse data spaces with very short documents such as code classes (Abadi et al. 2008).

3.4 Generating Multinomial Topics (the Term × Topic matrix)

Topics in topic modeling analysis are often described as multinomial distributions over words. Each word in the topic is assigned a weight based on the underlying statistical assumptions of the technique. Such weights are presumed to reflect the information value of specific words to the subject matter of the topic. In our analysis, we assign weights to each individual topic term according to the amount of mutual information it shares with other terms in the topic. This strategy draws on recent findings by Newman et al. (2010) and Lau et al. (2010), stating that the importance of a word to the topic's subject matter, as judged by humans, can be quantified based on its average semantic similarity with other words in the topic.

Following this strategy, in the proposed approach, the semantic importance of a code term is calculated as the mean pairwise NGD between that term w_i and all other terms in the topic t_k. This value, we refer to as the NGD_score, can be calculated as:

$$NGD\_score(w_i) = \frac{1}{|t_k|} \sum_{j=1}^{|t_k|} NGD(w_i, w_j) \;\; \forall i \neq j \tag{9}$$

The semantic weight of each term is then divided by the summation of the semantic weights of all terms in the topic for normalization. This ensures that each term is assigned a weight proportional to its value to the topic. The summation of these weights is equal to 1, thus producing our $term \times topic$ distribution. For example, using NGD_score, in the set of terms {`year`, `mmddyyyy`, `ddmmyy`, `month`, `format`, `date`}, extracted from the *iTrust* system, the term `date` acquires the highest average NGD_score with other terms, followed by the terms `month` and `format`. Similarly, in the topic {`font`, `width`, `style`, `bold`, `removes`, `helvetica`} from the *iText* system, the term `font`, followed by the term `style`, are dominating the topic by sharing the highest average mutual information with other terms in the topic. In both examples, dominant terms are the most descriptive words of the topic from a linguistic point of view, thus, can be chosen as labels for their topics.

3.5 Generating the Topic × Class Distribution

An important aspect of topic modeling analysis is to assign documents in the collection to the generated topics. In other words, generating the $topic \times document$ matrix. To generate such distributions, in the proposed approach, each topic is treated as an artifact. The similarity between each code class and each topic is then calculated using an adaptation of the text semantic similarity measure (TSS) proposed by Mihalcea et al. (2006). The original derivation of TSS combines information from word-to-word similarity and word specificity to calculate the similarity between short text paragraphs. Using these assumptions, in our

analysis, the semantic similarity between a code class d and a topic t can be described as follows:

$$sim(d,t) = \frac{1}{2}\left(\frac{\sum(maxSim(w_i,t) \times IDF(w_i))}{\sum IDF(w_i)} + \frac{\sum(maxSim(w_j,d) \times IDF(w_j))}{\sum IDF(w_j)}\right) \tag{10}$$

$maxSim(w_i, t)$ returns the NGD similarity score between the term w_i from the class d and its most similar term in the topic t. Similarly, $maxSim(w_j, d)$ returns the NGD similarity score between the term w_j from the topic t and its most similar term in the class d. $IDF(w)$ is the term's specificity, calculated as the number of classes and topics in the system divided by the number of classes and topics that contain the term w.

This formula, we refer to as TSS_NGD, combines the information value of terms within the class with the information value of terms within the topic. This ensures that very common terms in the class do not dominate the topic distribution of the class. It also ensures that most representative terms of the topic are given priority over less representative terms. After the similarity scores between a code class and all generated topics are calculated, such values are normalized (divided by the summation) to fit in the range [0 - 1], thus generating our $topic \times class$ distribution of the system.

3.6 Procedure

In summary, the proposed approach can be described as a formal procedure (Algorithm 1). This procedure is also depicted in Fig. 1 which shows the matrices, the calibration measures, and the optimization function, used to generate a topic distribution. For readability purposes, from this point onward, we refer to this approach as STM, **S**emantic **T**opic **M**odeling.

Algorithm 1 STM Procedure Summary

```
1: procedure
2:     Extract terms from source code
3:     Calculate NGD pairwise similarity between terms
4:     Cluster system terms using AL based on NGD
5:     Optimize Q (Eq. 8) to determine the number of topics
6:     Assign weights to topic terms based on NGD_score
7:     Assign code classes to topics based on TSS_NGD
8: end procedure
```

4 Evaluation

In the first phase of our analysis, we empirically identified the combination of distance measure (NGD), quality function (8), and clustering algorithm (AL), that generates the most semantically coherent clusters of software terms. We further suggested semantically aware strategies to produce multinomial distributions of software terms and topics. In this phase of our analysis, we are concerned with the quality of these topics. More specifically, do STM topics encapsulate meaningful themes, or domain concepts? To answer this question, we conduct a human experiment to assess the quality of STM topics in comparison to topics generated by conventional topic modeling techniques. In what follows, we describe our evaluation strategies, experimental setup, research questions, and main findings in greater detail.

4.1 Evaluating Topic Models

Accurate evaluation of topic modeling techniques is a challenging task (Wallach et al. 2009). This can be attributed to the fact that topic modeling is an unsupervised process. There is no *a priori* known topic distribution that the discovered topics can be compared against (Kuhn et al. 2007). Furthermore, there is a lack of automatic ways to assess the coherence, or usefulness, of generated topics. To overcome these challenges, in related NLP research, human subjects are often used to judge the quality of topics generated by various topic modeling techniques (Newman et al. 2010; Mimno et al. 2011). Humans are more capable of evaluating the exploratory goals of topics than solely relying on statistical measures which assess only a few aspects of generated models (Chang et al. 2009; Mei et al. 2007; Newman et al. 2010).

In our analysis, we adopt the popular *word intrusion* and *topic intrusion* tests to assess the quality of STM topics (Chang et al. 2009; Newman et al. 2010). These two tests are designed to capture the quality of generated topic distributions at the word and the topic levels as perceived by human judges. These tests can be described as follows:

- **The word intrusion test:** This test is designed to assess the semantic coherence of topics. A random sample of m topics from the generated topic space is first selected. For each selected topic, the n most important (probable) words are combined with an unrelated word (a word with very low probability) that is randomly selected from corpus. These $n+1$ words are shuffled and presented to a human judge. The judge is asked to identify the intruder word. In a semantically coherent topic, the unrelated word should be easy to identify. In a less coherent topic, identifying this outlier should be more difficult (Chang et al. 2009). For instance, in the topic ⟨ `emergency`, `emer`, `responder`, `insurance`, `health`, `accuracy` ⟩, from the *iTrust* system, it is obvious that the term `accuracy` is the intruder term. The precision of the model is calculated as the percentage of times the judge agreed with the model (was able to identify the intruder word).
- **The topic intrusion test:** This test is designed to measure the accuracy of the model in assigning topics to documents. After a topic model is generated for the corpus, a document is randomly selected from the collection. The top n topics from the topic distribution of that document, in addition to a random unrelated (very low probability) topic, are selected. These $n+1$ topics are shuffled and presented to the human judge along with the document. The judge's task is to identify the intruder topic, or the topic that does not reflect the document's subject matter. In a high quality topic distribution, the intruder topic should be easy to detect. The quality of the topic modeling technique can then be assessed as the number of times the human judge was able to detect the intruder topic. A high detection rate indicates that the model was able to generate topics that are more descriptive (more discriminative) of the document's main theme.

4.2 Experimental Systems

To conduct our word and topic intrusion tests, we use three software systems from different application domains. We considered two criteria for a system to be included in our experiment: ***a)*** the system has to be of a decent size that is representative of real life scenarios, and ***b)*** access to domain experts who are willing to participate in our experiment. Based on these criteria, our final dataset consisted of three systems, including:

- *iTrust*: A health care system that has been used in our lab as an experimental subject for the last three years. Two Masters students and one PhD student who have been working on *iTrust* for the period of two years were selected to participate in our experiment.
- *Apache_Ivy*: A subproject of the *Apache Ant* project that is designed to resolve Java projects dependencies in a flexible and simple manner. Three PhD students from two different research groups who have been using *Ivy* in their graduate research were selected to participate in our experiment.
- *BlueWallet*: A Java web service which provides users with options to plan their budgets and manage their personal finances. This system was contributed by one of our industrial partners. Three developers of *BlueWallet* participated in our experiment.

4.3 Experimental Baselines and Research Questions

We compare STM's performance against two other topic modeling techniques that are often used in source code analysis tasks. These techniques include:

- **LDA:** LDA (Blei et al. 2003) has been commonly employed in software engineering research. To implement LDA, we used JGibbLDA,[15] a Java implementation of LDA that uses Gibbs Sampling for parameter estimation and inference. As implied earlier, determining the optimal number of LDA topics for a software system is still an open research question (Panichella et al. 2013). In our analysis, we followed the guidelines in Lukins et al. (2010), Maskeri et al. (2008), and Bavota et al. (2014), using the heuristics $\langle \alpha = 1.0, \beta = 0.25, k \leq 100 \rangle$ for small systems ($\leq$ 100k LOC, $\leq$ 10k terms). We further set the number of topics (K) to 70, 65, 80, in *iTrust*, *Apache_Ivy*, and *BlueWallet* respectively. These numbers were selected based on our thorough experience with the internal structure of each system (expert judgment), including their folder structure, Object Oriented relations, and modular decomposition. In particular, all three experimental systems have been subjects of study in our previous work and one of the researcher is an original developer of the *BlueWallet* system. Using these configurations, LDA was executed several times (15 times for each system) and the best model (according to our expert judgment) was considered. The number of iterations was set to 1000, which is a quite common number of iterations to get well mixed samples. If the number of iterations is too low, generated distributions are often random and unstable. However, a large number of iterations allows the sampler to reach a relatively steady state at which the generated distributions tend to be more stable (Binkley et al. 2014).
- **LSA-Semantic Clustering (LSA-SC):** This approach is based on Kuhn et al. (2007) software semantic clustering technique. This technique exploits the textual information of source code using LSA and AL clustering to cluster software artifacts based on their similar vocabulary (i.e., identify clusters of linguistic topics in source code). To implement LSA-SC, the same procedure for generating STM topics is used. In particular, the same calibration equations used in STM ((8), (9), and (10)) are used. However, LSA is used instead of NGD as the term pairwise semantic similarity measure. The dimensionality reduction settings identified earlier (Table 2) are used to calibrate LSA-SC.

[15] http://jgibblda.sourceforge.net/

Given our experimental setup and performance tests, we formulate the two following research questions:

- ***RQ1***: At the term level, are STM topics more semantically coherent than LDA and LSA-SC topics?
- ***RQ2***: At the topic level, are STM topics more descriptive (discriminative) of the system artifacts than LDA and LSA-SC topics?

4.4 Execution and Results

To conduct the word (code term) intrusion test (i.e. answer our first research question), three topic models (LDA, LSA-SC, and STM) were generated for each of our experimental systems. Ten topics were then randomly selected from each topic model generated for each system, resulting in a total of thirty topics. The four most probable terms from each topic, along with a randomly selected term from the system, were then selected. These five term lists from each of our experimental systems were shuffled and presented to each of our study participants from that system. The participants were asked to identify the term in each list that does not belong to the topic. To minimize the experimental bias, the random topic selection process is automated. In particular, the .Net `random.Next(1,K)` method is used to pick a topic from the K topics generated for the system. The task was demonstrated to the participants by the researchers using sample topics prior to the experiment. No time constraint was enforced.

The results of the word intrusion test are shown in Fig. 4. The box plots in the figure show the aggregated performance of our study participants in each of our experimental systems. In general, our participants were more successful in detecting the intruder terms in STM topics, thus answering ***RQ_1***. To test the significance of the results, each experimental system (*iTrust*, *Apache_Ivy*, and *BlueWallet*) served as a between groups factor, while the topic modeling technique served as a within groups factor in a 3x3 mixed model ANOVA. The omnibus model revealed a significant effect of topic modeling technique $F(2,12) = 10.62$, $\boldsymbol{p} = .002$, η_p^2=0.64, while no effect was observed between the three different software systems $F(2,6) = 0.26$, $\boldsymbol{p} = 0.78$, η_p^2=0.08. Likewise the interaction between the two variables failed to reach significance $F(2,6) = 0.16$, $\boldsymbol{p} = 0.85$, η_p^2=0.05. These results confirm that the large differences in intrusion detection were reliable and consistent across our software

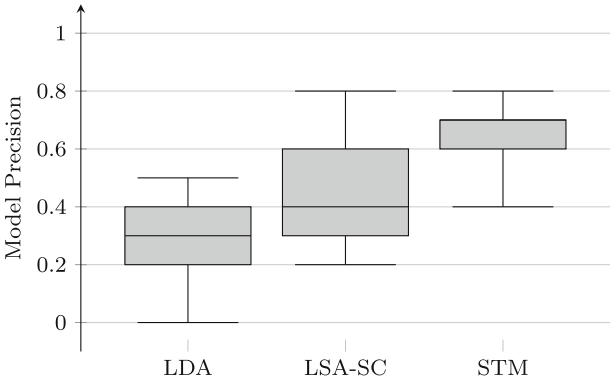

Fig. 4 Word intrusion test's results aggregated per model across all our experimental systems

systems. A planned comparison was also conducted to test the hypothesis that STM (the top performing method) was superior to LSA-SC (the second best method) and showed a significant difference t(8)=2.25, $\boldsymbol{p}$ = .028, d=0.76. A post hoc comparison also showed that LSA-SC outperformed LDA (t(8) = 3.41, $\boldsymbol{p}$ = .009, d=1.175).

To conduct the topic intrusion test (i.e. answer our second research question), three topic models (LDA, LSA-SC, and STM) were generated for each of our experimental systems using the configuration settings identified earlier. Ten classes were randomly selected from each system. Three intrusion test units were created for each class. Each test unit included the class, the two most probable topics of the class, and one unrelated topic randomly selected from the class's topic distribution. Given the homogeneity of code classes, two topics were found to enough to describe the class's functionality (De Lucia et al. 2012). This set of three topics was shuffled and presented to the participant. Each topic in the pool was represented by its eight most probable terms. Eight words were found to be convenient enough to describe a topic (Chang et al. 2009; Newman et al. 2010; Mimno et al. 2011). The study participants from each system were then asked to discard a single topic which they thought did not sufficiently capture the class's subject matter (the intruder topic). In particular, the following question was asked: *Which of the following topics do you think does not represent or describe this class?* We elaborated by emphasizing that we were looking for topics that were more expressive of the class's functionality, including the methods it encapsulates and its overall role in the system. The study participants were allowed to view any part of the system (e.g., classes in different packages or folders) to help them make more informed decisions.

In our topic intrusion test, we minimized carryover effects by ensuring each class and each topic appears in every serial position and that a class was never judged immediately before or after the same class. Participants only made two judgments on each class, which minimized learning effects and fatigue factors. For example, assuming a sample system $\boldsymbol{A}$ with three participants $\{S_1, S_2, S_3\}$, three classes $\{C_1, C_2, C_3\}$, and three topic models $\{T_a, T_b, T_c\}$ generated using three different topic modeling techniques. For each class C_i, three topic intrusion tests are created, one for each topic distribution T_j generated for the class. This test unit can be described as $C_i T_j$. Therefore, the list of test units for the system $\boldsymbol{A}$ includes: $\{C_1T_a, C_1T_b, C_1T_c, C_2T_a, C_2T_b, C_2T_c, C_3T_a, C_3T_b, C_3T_c\}$. Participants judge each class twice and each topic model twice, following a pseudo Latin-Square design. For the sample system $\boldsymbol{A}$, this process is depicted in Table 3.

The results of our topic intrusion experiment are shown in Fig. 5. Logistic regression analysis was conducted to predict the ability of participants to detect intruder topics in ten different classes created by the three distinct topic modeling techniques (STM, LDA, LSA-SC) for our three different experimental systems. Predictors included the ten classes for each system, three participants, and three topic modeling techniques, all entered with contrast codes. The full model was a significant improvement over the constant only model, indicating the predictors collectively distinguished between where the intruder topics could

Table 3 Pseudo Latin-square design for the topic intrusion test. Each participant executes only two intrusion tests per class

Participant	Topic intrusion test units					
S_1	C_1T_a	C_2T_c	C_3T_b	C_1T_c	C_2T_b	C_3T_a
S_2	C_2T_b	C_3T_a	C_1T_c	C_2T_a	C_3T_c	C_1T_b
S_3	C_3T_c	C_1T_b	C_2T_a	C_3T_b	C_1T_a	C_2T_c

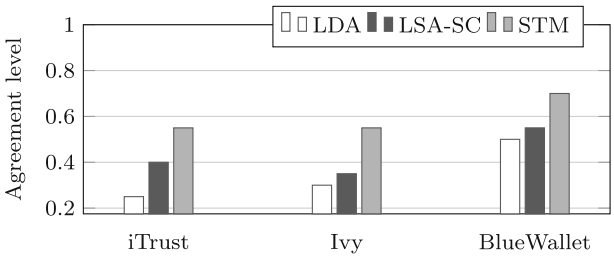

Fig. 5 Topic intrusion test's results averaged for all participants over each model (LDA, LSA-SC, STM) in each system

be detected from those where it could not (χ^2 = 59.66, $\boldsymbol{p}$ = 0.011, 37 df). The relationship between prediction and grouping was moderate (Nagelkerke's **R** of 0.377). Moreover, of particular interest was the difference between methods, evaluated by contrasting STM *vs.* LSA-SC and LDA *vs.* LSA-SC (LSA-SC was associated with an intermediate level of intrusion detection). For the STM *vs.* LSA-SC contrast, β = 0.906, $\boldsymbol{p}$ = 0.04, indicating that STM produced significantly greater intrusion detection than did LSA-SC. LDA (β = -0.486, $\boldsymbol{p}$ = 0.30) was not statistically worse than LSA-SC, though the trend was in that direction. Accordingly, to answer ***RQ_2***, STM was superior to both other methods, which did not differ significantly from one another.

5 Discussion

The results of our human experiment confirm previous observations that humans conceptualize documents in such a way that is different from statistical topic modeling techniques (Chang 2010). This conceptualization seems to be effectively captured by the distributional cues of terms in the system (Newman et al. 2010), indicating that shallow text properties, such as term co-occurrence, are actually sufficient to capture the textual semantic patterns in source code.

In the design of STM (Fig. 1), we focused on minimizing calibration effort. In other words, simplifying the process of implementing the approach and producing results while keeping the application cost as low as possible. This design constraint is motivated by recent findings in software engineering research which revealed that a tool would be deemed as *"not needed"* by practitioners if it has a high operating cost (Lo et al. 2015). In particular, while classical topic modeling techniques such as LDA, LSA, and PLSI require calibrating several parameters (α, β, K, number of iterations), STM supports a self optimizing procedure with relatively low calibration effort. STM automatically looks for a solution in the search space (cluster hierarchy) guided by a quality objective function that captures the semantic coherence of generated topics. This ensures that terms of each cluster exhibit a high degree of meaning overlap, which is a property of well written texts. Furthermore, STM does not break the *bag of words* assumption, therefore, no extra processing is required. However, it is important to point out that, just like other topic modeling techniques, STM is prone to suboptimality. This can be attributed to the fact that HAC algorithms follow a

greedy strategy when deciding which two clusters to merge on a given step. Consequently, they might settle for suboptimal solutions (Zhao and Karypis 2002).

Another characteristic of STM is its stability. This quality derives from the stability of the underlying clustering strategy which does not depend on randomization steps inherent in other clustering algorithms (e.g., K-means). From a practicality point of view, this gives STM an advantage over probabilistic techniques such as LDA and PLSI. For instance, LDA comes with no guarantee that different runs over the same source material will generate similar distributions. As mentioned earlier, exact inference is intractable, thus the generated topics for the same training set might vary depending on the initial sampling process (Potapenko and Vorontsov 2013; Koltcov et al. 2014; Wallach et al. 2009). In fact, there is no practical way of knowing exactly how many iterations are required to reach a stationary state that represents the actual topic distribution over the corpus (Binkley et al. 2014). In contrast, if the data space remains unchanged, STM should generate identical topic distributions over multiple runs.

In addition to its stability, STM adopts a crisp (non-overlapping) clustering mechanism to produce topics. In other words, each term can belong to one topic only. This constraint is enforced to ensure the lexical sparsity of generated topics. From a linguistic point of view, sparsity is a highly desired property of topic distributions. Sparse topics are more representative (semantically coherent) of the data as they often carry more distinctive information. This can be crucial for applications such as document classification and retrieval (Steyvers and Griffiths 2007; Than and Ho 2012) and for tasks at which micro changes need to be detected. However, crisp classification might suffer when a term is used in different contexts. For example, assume a developer used the term `compute` in two different classes with different topics (e.g., `computeSalary` and `computeAreaUnderCurve`). Ideally, this term should be assigned to two different topics. However, due to our non-overlapping clustering strategy, the term `compute` will be assigned to one topic only (the topic with which it shares more semantic relations). A potential work-around for this problem is to allow words with strong relations to multiple topics to be assigned to more than one topic.

In terms of space complexity, STM works with one principal matrix of raw co-occurrence data to create topics. This matrix is symmetrical (i.e., its upper and lower triangles are identical) which reduces the space requirements by half. Given the high ratio of non-co-occurring terms often found in software systems, the size of this matrix can be reduced even more. Furthermore, given that we adopt a non-overlapping clustering approach to create topics, the space requirements for storing the STM topic distribution of a system is $O(N)$, where N is the number of terms in the system. In addition, STM uses a text similarity measure for assigning documents to topics, thus, zero similarity is possible. However, in techniques such as PLSI and LDA, each term in the corpus appears in each topic, thus requiring $O(KN)$ space requirements, where K is the number of generated topics. Similarly, to store the $topic \times document$ matrix, LDA requires $O(DK)$ space where D is the number of artifacts in the system. Such representations are extremely dense and consume more memory space (Than and Ho 2012).

In terms of limitations, we acknowledge that STM exhibits a relativity high time complexity, a characteristic it shares with most topic modeling techniques. For instance, optimal LDA solutions are sparse and rarely found and finding the maximum *a posteriori* assignment of topics to words in LDA (exact inference) is NP-hard (Sontag and Roy 2011). Therefore, approximate inference models are adopted to solve the problem. Even with this compromise, reaching a convergence point is frequently time consuming. For example,

Gibbs sampling requires 500-2000 iterations to reach a stable state (Biggers et al. 2014; Binkley et al. 2014; Panichella et al. 2013; Newman et al. 2011; Thomas 2011). In comparison, STM acquires a time complexity of $O(N^3)$, where N is the number of terms in the system. The performance bottleneck in STM is caused by the underlying term clustering step. The complexity of HAC clustering algorithms is $O(N^2)$. The self optimization step requires $O(N)$ time since the algorithm has to climb up the N levels of HAC dendrogram until an optimal cut point is found. This raises the overall running time to $O(N^3)$. Our expectation is that such complexity can be reduced using optimal implementations of the underlying clustering algorithm.

It is important to point out that, while the proposed approach adopts a hierarchical clustering strategy, it is fundamentally different from hierarchical topic models (hLDA) (Blei et al. 2003). hLDA is an extension of LDA that organizes the topics in a document's topic distribution in a tree, thus determining the relationship between topics and their level of abstraction. More specifically, a document in the collection is assumed to be generated by the set of topics along a path of the topic tree. In such models, the quality of the generated distribution depends on the structure of the topic tree. To infer this structure, hLDA applies a nested Chinese restaurant process (NCRP)—a widely used method in Bayesian nonparametric statistics. hLDA has the principal advantage of allowing the model parameters and its structure to be automatically adapted as more data is observed. However, the performance of the model is very sensitive to the prior information, such as number of levels (depth of the tree) and number of topics in each level. Furthermore, such models tend to fail when the data is sparse, which is an inherent characteristic of source code.

6 Case Study: Analyzing Code Evolution using STM

Topic modeling has been recently used to analyze code evolution (Linstead et al. 2008; Thomas et al. 2010). The main assumption is that programmers name their code fields and write their comments in such a way that reflects their understanding of the system (Deißenböck and Pizka 2005). Therefore, analyzing such information through linguistic models is expected to provide a unique insight into the evolution of code, revealing changes that often go unnoticed by other methods (e.g., bug fixes, refactoring activities, and feature addition). Understanding and modeling the dynamics of code evolution helps practitioners and researchers to diagnose and reverse the symptoms of code aging. For instance, by tracing emergent problems to their original causes in the archive of system releases, accurate predictions can be made about future bugs (Chidamber and Kemerer 1994).

In this case study, we use STM to track and analyze change in the mobile application *Flym*.[16] *Flym* is an open source Android application that fetches user favorite websites and blogs and displays them in a mobile-optimized way. We sampled and analyzed ten releases of *Flym* using STM. The characteristics of these releases are shown in Table 4. STM is then used to generate a topic distribution for each release of *Flym* (stemming was not applied to reserve the naturalness of words). Table 4 shows the number of topics generated for each release. We then analyze the changes in the topic distributions over the multiple releases of the system. In particular, we start our analysis by observing the changes in the assignment of individual topics over time. Topic assignment reflects the extent a topic is manifested in a specific release (Linstead et al. 2008) and is calculated as the number of artifacts assigned

[16] https://github.com/FredJul/Flym

Table 4 *Flym's* Releases

No.	release	kloc	comments (kilo)	No. class	No. topics
1	v1.0.8	7,618	1,298	34	21
2	v1.1.0	7,792	1,297	34	26
3	v1.2.0	8,394	1,338	36	26
4	v1.3.0	8,369	1,199	34	24
5	v1.4.0	8,868	1,288	37	27
6	v1.5.0	9,937	1,401	40	27
7	v1.6.0	9,917	1,535	46	30
8	v1.7.0	9,917	1,535	46	31
9	v1.8.0	9,917	1,535	46	30
10	v1.9.0	9,917	1,535	46	30

to the topic at each release, divided by the total number of assignments over all releases. A document is considered assigned to the topic if the topic appears as one of the top three most probable topics of the document.

A subset of the most dominant topics over the multiple releases of *Flym* is shown in Table 5. A topic that shows an interesting behavior is {`fullscreen, cancel, flag, window, visible`}. To track the evolution of this topic, we manually track the topic that has the term `fullscreen` as its most probable term. Figure 6 shows the change in this topic's assignment over the ten releases of *Flym*. In Table 6, we also report the five most probable terms of the topic at the different releases of *Flym*. Our first observation is that this topic first appears in the release v.1.2.0 of the system. A close examination of the code reveals that the feature *fullscreen* first appears at release v1.2.0. This feature is implemented in the class `EntryActivity.java` through the function `onClickFullscreenBtn(View view)`. At release 1.4.0, the class `EntryActivity.java` drops its assignment to the topic. The topic now dominates the two classes `EntryFragment.java` and `BaseActivity.java`. This suggests that the *fullscreen* feature has been moved to these two classes. A plausible scenario is that a combination of EXTRACT CLASS and MOVE METHOD refactorings have taken place. Starting from release 1.7.0, the *fullscreen* feature now spreads over six classes (more cross-cutting), causing a spike in the topic's assignment.

Examining the most probable words of the topic itself shows that the topic becomes more well defined as the system progresses. The five most probable words of the topic when it first appears are {`fullscreen, cancel, flag, mobilized, visible`}. We notice that at release v1.4.0 of the system, the term `toggle` has found its way into the topic's

Table 5 A subset of *Flym's* most dominant topics

Topic
{fullscreen, cancel, flag, mobilized, visible}
{view, drag, position, icon, pixel}
{merge, publish, provide, retrieve, red}
{graphic, bitmap, clock, drawable, image}
{widget, size, font, app, activity}

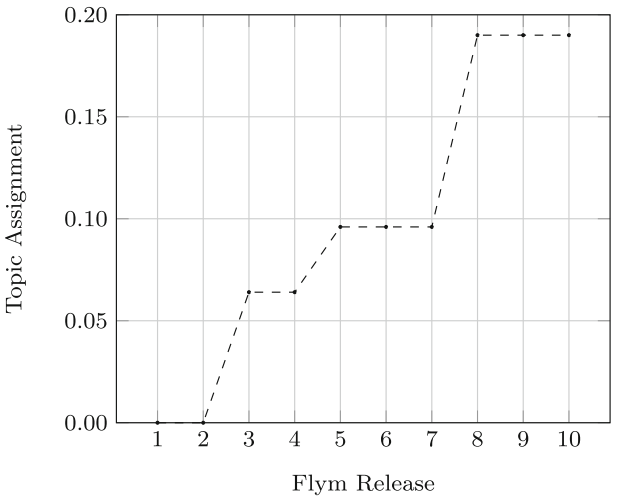

Fig. 6 Change in the document assignment of the topic {`fullscreen, cancel, flag, mobilized, visible`} over the multiple releases of *Flym*

most probable words. An examination of the code reveals that a RENAME METHOD refactoring has taken place. More specifically, the method `onClickFullscreenBtn(View view)` has been renamed to `toggleFullScreen()`. We also notice that at release v1.7.0 of the system, the term `immersive` starts gaining more weight. Examining the code reveals that the code for the *immersive mode* has been added the class `BaseActivity.java` to work with the *fullscreen* feature of the system.

In summary, our case study shows that STM was able to detect changes in the system to a great extent. These changes reflected distinctive maintenance activities such as feature addition and refactoring. The sensitivity of STM to fine grained changes can be

Table 6 The evolution of the topic `fullscreen` over ten releases of the system *Flym*

Release	Top five terms	Classes
v1.0.8 v1.1.0	NA	NA
v1.2.0	{`fullscreen, cancel, flag, mobilized, visible`}	EntryActivity.java
v1.3.0	{`fullscreen, cancel, flag, window, mobilized`}	EntryActivity.java
v1.4.0 v1.5.0 v1.6.0	{`fullscreen, toggle, cancel, flag, window, visible`}	BaseActivity.java EntryFragment.java
v1.7.0	{`fullscreen, toggle, cancel, flag, immersive, visible`}	BaseActivity.java EntryFragment.java PrefUtils.java
v1.8.0 v1.9.0	{`fullscreen, toggle, flag, immersive, cancel, video`}	BaseActivity.java EntryFragment.java EntryView.java

particularly useful for systems that evolve in a rapid pace with small increments (e.g., mobile applications).

7 Threats to Validity

The study presented in this paper has several limitations that might affect the validity of the results. Internal validity refers to confounding factors that might affect the causal relations established in the experiment (Dean and Voss 1999). A potential threat to the proposed study's internal validity is the fact that human judgment is used as a basis to assess the quality of generated topics. This might result in an experimental bias as humans tend to be subjective in their judgment. However, as stated earlier, at the current state of topic modeling research, humans are often used as the ultimate judge of topics' expressiveness (Chang et al. 2009; Newman et al. 2010). Therefore, these threats are inevitable, but they can be partially mitigated by using a diverse subject sample including domain experts at different levels of expertise. We further acknowledge the fact that the configuration settings and calibration heuristics used to tune LDA might affect the internal validity of the study. However, as mentioned earlier, there is still no robust method available to determine such settings, and thus, domain expert advice is still the approach adopted in most LDA based studies (Andrzejewski et al. 2007; Gethers and Poshyvanyk 2010; Maskeri et al. 2008).

The class granularity level adopted in our analysis can also be considered as a threat to the internal validity. In particular, different granularity levels might considerably change the behavior of our approach. However, since we deal with Object Oriented systems, each class supposedly encapsulates one functionality and thus can be treated as a separate document. The decision to consider a class level is also based on the observation that higher granularity levels (e.g., a code snippet or a method) do not provide sufficient window size to collect co-occurrence data, or even provide sufficient context for methods such as LSA and LDA to work (Bullinaria and Levy 2007). On the other hand, at the package level, the window size is so big that co-occurrence cues become meaningless, thus increasing the likelihood of irrelevant and misleading information and depriving NGD, LSA, and LDA of context. Nonetheless, we believe that a future study, dedicated to investigating the effect of granularity level, or even other parameters such as the size of the system, on the performance of the proposed approach, is necessary to further confirm our observations.

Threats to external validity impact the generalizability of results (Dean and Voss 1999). In particular, the results of our experiment might not generalize beyond the specific experimental settings used in this paper. A potential threat to our external validity stems from the systems used in our experiment. Open source systems are likely to exhibit different characteristics from proprietary systems. We also note that our systems are of medium size, which may raise some scalability concerns. We believe that using multiple software systems, drawn from different application domains, helps mitigate these threats. Another limitation is the fact that these systems are Object Oriented systems that are developed in Java, leaving the question of how well our findings will generalize over systems developed in different programming languages, or over structured code, unanswered. Another external validity threat might stem from the case study we used to demonstrate the operation of our approach. Our case study was conducted using a limited number of releases of a single software system with a relatively small size. Therefore, it is still unclear to us whether our findings in this study will hold for larger numbers of releases or different size systems.

8 Related Work

In this section, we review four categories of research related to our work in this paper, including:

- Applications of topic modeling in software engineering
- Software clustering and categorization
- Estimating semantic similarity in source code
- Evaluating topic modeling techniques

8.1 Applications of Topic Modeling in Software Engineering

Topic modeling has been applied to a broad range of applications in software engineering. In what follows, we review related research on topic modeling applications in several basic software engineering activities.

8.1.1 Code Analysis and Comprehension

In source code analysis, topic modeling techniques are mainly used as a means to reduce the complex textual content of software systems down to more coarse grained, and supposedly easier to comprehend and synthesize, representations (i.e., topics). For instance, Maskeri et al. (2008) suggested a human assisted approach based on LDA to extract business domain topics from source code. The main objective is to help newcomers comprehend the functionality of large legacy software systems. Tian et al. (2009) employed topic modeling to automatically categorize software systems in open source repositories. The authors used LDA to extract topics across multiple software systems. These topics were clustered based on their word distributions into meaningful categories that represent distinct types of software. Savage et al. (2010) introduced TopicXP, an Eclipse plugin for extracting, analyzing, and visualizing unstructured information in source code identifiers and comments using LDA. TopicXP is intended to help developers working on maintenance tasks to comprehend the system and find relevant results more effectively. Gethers et al. (2011) presented CodeTopics, an Eclipse plugin that is designed to capture and visualize the similarity between source code and other software artifacts such as requirements and design documents. The main objective is to help developers identify requirements that have not been implemented yet and help newcomers to comprehend the code more effectively by allowing them to view the topics that different artifacts relate to. Thomas et al. (2010) and Linstead et al. (2008) used topic modeling to model and understand software evolution. The main assumption is that milestone change activities in software (e.g., bug fixes, refactoring) are reflected in the topic distribution of the system. For instance, a topic that emerges on the system's release time line often represents an integration point of a new functionality.

8.1.2 Software Artifact Retrieval

Topic modeling techniques have been used as IR mechanisms to match and retrieve different types of software artifacts. The main assumption is that artifacts exhibiting topical similarity are highly likely to be related. Such relations can be utilized to provide automated support for tasks such as automated requirements traceability, code retrieval, and bug localization. For instance, Asuncion et al. (2010) proposed a tool suite that utilizes LDA to

automatically capture traceability links during software development and estimate a probabilistic topic model over captured artifacts. The authors demonstrated through examples how topic modeling can provide support for prospective traceability and help developers to find relevant code. Oliveto et al. (2010) used LDA as a retrieval approach to trace requirements to their implementations. The authors compared the performance of LDA with other IR methods often used in traceability research, including the Jensen-Shannon Model (JSM), Vector Space Model (VSM), and LSI. The results showed that LDA was able to capture a set of unique links that other methods failed to capture. Similar observations were made in Abadi et al. (2008) and Mahmoud and Niu (2015).

Lukins et al. (2008) used LDA and LSI to match bug reports with their relevant buggy source code fragments. The effectiveness of the proposed approach was evaluated using three case studies. The results showed that LDA achieved a significantly better performance than LSI for most bugs. Nguyen et al. (2011) observed similar results using *BugScout*, a tool that uses LDA to locate buggy source code. *BugScout* considers past bug reports, in addition to source code comments and identifiers, to identify technical topics representing a source code document. Evaluating *BugScout* over four different projects showed that it recommended at least one buggy file correctly in 45 % of the cases with a ranked list of ten files.

8.1.3 Software Testing and Debugging

Andrzejewski et al. (2007) proposed using LDA to support statistical debugging tasks in software systems. Each run (execution trace) of the program is modeled as a document. Specific segments of code that are triggered during each run are modeled as words. Words generated from successful runs of the program are modeled as usage topics, while words generated from failed runs are modeled as mixtures of bug topics and usage topics. Bug topics are then used to explain the differences between successful runs and failed runs. The proposed approach was evaluated using multiple programs. The results showed that LDA was able to produce bug topics that were highly correlated to true bugs and to identify bug topics that would otherwise remain undetected.

Thomas et al. (2014) used topic modeling in static black box test case prioritization. The main objective is to identify test cases that should be executed first to maximize the fault detection rate of the test suite. The authors' proposed approach exploits the textual content of test cases (identifiers, comments, and string literals) to approximate the functionality of each test case using topics. Generated topic distributions of test cases are then compared to automatically assign higher priority to test cases with minimum overlap. The authors used multiple open source systems to evaluate the proposed approach. The results showed that it outperformed other existing static and dynamic test case prioritization techniques.

8.1.4 Code Modularization

Topic modeling has also been used to assess the modular structure of the code, or the degree to which elements of a module belong together (Bieman and Kang 1995). For instance, Bavota et al. (2014, 2014) used Relational Topic Models (RTM) to help developers improve the modularization of their software systems. Their proposed approach analyzes the latent topics in source code as well as structural dependencies between code entities to identify and recommend refactorings (e.g., Move Class) that enhance code modularity. Liu et al. (2009) proposed Maximal Weighted Entropy (MWE), a topic modeling based approach for

measuring class cohesion. MWE exploits the textual content of code classes through LDA and entropy measures to quantitatively assess their conceptual cohesion and predict their fault proneness. MWE was evaluated using two case studies. The results showed that MWE was able to capture different aspects of class cohesion compared to several existing cohesion measures.

8.1.5 Concern location

Baldi et al. (2008) proposed a new theory of aspects in Aspect Oriented Programming based on topic modeling. This theory suggests that concerns in source code are latent topics that can be automatically extracted using statistical topic modeling techniques. Using LDA, the authors managed to identify topics that emerged as general purpose aspects across multiple projects, as well as project specific concerns. Chen et al. (2012) used LDA to approximate concerns present in source code as topics. These topics were leveraged using multiple static and historic metrics that use the defect history of topics and their defect density to explain the defect proneness (i.e., quality) of different source code entities. Multiple case studies were conducted to evaluate the proposed approach. The results showed that different topics exhibit different levels of defect proneness, where defect prone topics tend to remain so over time.

8.1.6 Mining Free Text Software Artifacts

In addition to source code, topic modeling techniques have also been applied to analyze and synthesize other forms of software relevant text, such as developers discussions on online development forums and open source platforms. The main objective is to gain insights into the common trends of the development community. For instance, Barua et al. (2014) used LDA to automatically discover the topics present in developers interactions on Stack Overflow. The analysis showed that developers use Stack Overflow to discuss a wide range of topics from jobs to version control systems to code syntax. The analysis revealed that questions in some topics often lead to discussions in other topics. Hindle et al. (2015) investigated the high level traceability of topics extracted from software requirements and bug reports to version control commits using topic analysis. The authors reported that generated topics made sense to practitioners and often matched their perception of what activities took place at that time. Neuhaus and Zimmermann (2010) used topic models to analyze information security vulnerability reports in the Common Vulnerability and Exposures (CVE) database. The authors used LDA to automatically find the prevalent topics in the CVE and identify the emerging security trends in their data.

8.1.7 Calibrating Topic Models in SE tasks

Several topic modeling studies in software engineering have focused on identifying methods to calibrate topic modeling techniques in a variety of software engineering tasks. The majority of these methods use machine learning techniques to identify combinations of configurations that achieve acceptable performance levels in specific software engineering tasks. For instance, Lohar et al. (2013) used Genetic Algorithms (GAs) to search through a feature model of traceability configurations (or *features*) to identify the combination of features that achieves the most accurate results over a reference set of traceability links. The proposed approach is supervised. It relies on a training set of source and target software artifacts to guide the search process.

Panichella et al. (2013) also used Genetic Algorithms in an attempt to identify near optimal hyperparameters of LDA that achieve acceptable performance across various software engineering tasks. The main assumption is that proper LDA configurations should produce high quality clusters of software artifacts. The proposed approach was evaluated over multiple software engineering tasks, including requirements traceability, feature location, and software artifact labeling. The results showed that GAs led to higher accuracy levels compared to alternative methods.

In a more recent work, Panichella et al. (2016) proposed GA-IR, a technique that uses GAs to automatically configure and assemble an IR process for specific software engineering tasks. Similar to the work in Panichella et al. (2013), the proposed approach is unsupervised in the sense that it uses GAs to approximate configurations (e.g., term extraction, stop-word removal, stemming, indexing and IR methods calibration) that maximize the overall quality (fitness function) of the retrieval process as measured by the quality of generated software document clusters. GA-IR was evaluated over traceability link recovery and duplicate bug reports detection tasks. The results showed that IR processes assembled by GA-IR significantly outperformed those assembled according to existing approaches.

8.2 Software Clustering and Categorization

Related to our work is *Semantic Clustering*. Proposed by Kuhn et al. (2007), semantic clustering is a technique that uses Latent Semantic Indexing (LSI) and Average Linkage (AL) to identify linguistic topics in source code. The proposed approach exploits the linguistic information found in source code to cluster software artifacts based on similar vocabulary. Semantic links between generated clusters are then captured to automatically illustrate how topics are distributed over the system. Semantic clustering shares several processing steps and exploratory objectives with STM. However, it is calibrated through several manually specified thresholds (the optimal number of clusters), and heuristics (LSI parameters) to achieve near optimal results. STM, on the other hand, is a fully automated process that finds optimal solutions by maximizing a predefined quality function.

Mancoridis et al. (1999, 2006) proposed *Bunch*, a clustering tool designed to recover software systems structures. Bunch tries to find the number of optimal clusters in a software system by mapping the system's artifacts, based on dependency relationship between program entities, into a Module Decomposition Graph (MDG). This graph is then solved by finding a good partition that best separates its nodes. In particular, the problem is treated as an optimizing problem with an objective quality function (Modularization Quality). Hill-climbing and Genetic algorithms are used to find graph partitions that optimize the quality function. An important limitation of this approach is that it can be intractable as the number of possible graph partitions grows exponentially with the number of nodes, thus the solutions found are typically suboptimal.

Tzerpos and Holt (2000) proposed ACDC, a comprehension driven clustering algorithm that exploits different patterns in software systems, such as the system directory structure, the procedures and variables in source files, and class body header relations, to identify clusters in the system. Comparing ACDC results against manually prepared authoritative decompositions showed that ACDC produced meaningful partitions suitable for program understanding. ACDC is different from STM in the sense that it utilizes structural information rather than linguistic information to create clusters. Furthermore, ACDC is targeted toward comprehension, while STM is designed as a general purpose technique that can serve multiple analytical tasks, including program comprehension.

Anquetil et al. (1999) conducted a comprehensive analysis of software clustering as a technique for software remodularization. The authors analyzed the performance of several hierarchical clustering algorithms and different quality measures to compute coupling between software entities. The results showed that non-formal features of code (e.g., comments and identifiers' names) can be as effective as structural information to classify software. The results also showed that Complete Linkage achieved the best results in terms of producing cohesive clusters. Average Linkage, on the other hand, produced the best coupling, while Single Linkage failed to achieve satisfactory performance. Our findings in this paper correspond to these observations regarding the family of HAC algorithms and the information value of the non-formal features of the code.

8.3 Estimating Semantic Similarity in Source Code

Sridhara et al. (2008) examined the performance of six different similarity measures in estimating the semantic similarity between individual code terms in a software system. These measures include information, gloss, and path based methods that utilize the hierarchies of WordNet to establish similarity relations between words. Comparing these methods against a set of manually identified semantically similar code term pairs revealed that using external sources of linguistic knowledge in the context of source code can yield random results.

In our previous work (Mahmoud and Bradshaw 2015), we conducted a systematic analysis of a series of semantic relatedness methods to capture semantic relatedness between source code terms. These methods included: LSA (Deerwester et al. 1990), NGD (Cilibrasi and Vitanyi 2007), PMI (Turney 2001), and the set of thesaurus based methods used in Sridhara et al. (2008). We used a dataset of human generated similarity data to compare the performance of these methods. The results showed that NGD, utilizing corpus based co-occurrence patterns of code terms, was the most accurate in capturing the notion of similarity between code terms.

Tian et al. (2014) leveraged co-occurrence information extracted from *StackOverflow* to estimate semantic relations between software terms. The main objective was to automatically construct a taxonomy of software engineering words. Technically, the proposed approach represents each term as a feature vector of its co-occurrence information with other terms in the system. A derivation of PMI is then used to measure the similarity between terms' vectors. The authors evaluated their approach against a WordNet based approach using a set of manually prepared software words. The results showed that the proposed approach has achieved significant improvement in accuracy.

Yang and Tan (2014) analyzed the contextual information of source code to automatically infer semantically related terms in a software system. The proposed technique uses the Longest Common Subsequence (LCS) to find the longest overlapping term subsequences between two term sequences extracted from comments and method names. The primary assumption is that if two terms or phrases are used in the same context, then they are likely to be semantically related. This approach makes similar assumptions to the approach proposed in our paper. However, it can be more computationally expensive as it requires operations such as parsing comments and code, creating words sequences, and calculating LCS. In addition, the quality of extracted similarity relations depends on the availability of a large amount of comments and meaningful identifiers.

Howard et al. (2013), proposed an automatic technique for mining semantically similar words from several software applications based on extracting action verb pairs from methods' signatures and their header comments. Such pairs are assumed to describe the computational intent of the method, thus they are potentially semantically related. The

authors have reported high correlation levels to manually extracted related words from software systems. However, similar to Yang and Tan (2014), this approach relies heavily on the availability of descriptive comments and method signatures. In addition, it depends on the existence of verb-noun (action) sequences in the comments.

8.4 Evaluating Topic Modeling Techniques

Initial work on human based evaluation of topic models was described by Chang et al. (2009). In particular, a topic is selected from a generated topic model. The five most probable words of the topic, along with a random word from the corpus (a high probability word in another topic), are randomized and presented to a human judge to identify the intruder word. An unrelated word should be easily observable in a semantically coherent topic. Similarly, human participants are presented with a random document's title and a snippet. The three most probable topics assigned to this document, along with an intruder topic (low probability for the selected document), are randomized and presented to the human judge. If the model is correct, an unrelated topic should be easily identified.

Newman et al. (2010) proposed a measure to estimate the coherence of a topic as the sum of pairwise distributional similarity of its most probable words. To conduct their experiment, topic distributions were generated for two document collections. Nine human annotators were then asked to judge the coherence (usefulness) of samples of the generated topics. Spearman rank correlation was calculated between the human generated rankings of coherence and the different automated coherence measures. The results showed that PMI, calculated based on information extracted from Wikipedia, achieved the highest correlation with human judgment.

Mimno et al. (2011) proposed an automated corpus based evaluation metric for identifying flawed topics. The authors' main objective was to identify semantic problems in topic models without the need for external reference corpora such as Wikipedia. In particular, a golden standard of human annotated topics was created over a large dataset of research articles. Average similarity between each topics' most probable words was then calculated based on the co-occurrence of these words in the corpus. This corpus based metric was found to achieve high levels of correlation with human judgment of topics' quality.

Stevens et al. (2012) suggested a new metric of topic coherence to evaluate full topic distributions rather than individual topics. In particular, the set of discovered topics over a corpus was treated as generalized semantics describing individual words in the corpus. These words were then represented as feature vectors based on their distributions over the topics. The main assumption was that similar words are likely to be represented by similar topics. Pairwise cosine similarity of word pairs was then compared to manually generated word similarity information. A higher correlation implies a higher quality topic model.

9 Summary and Future Work

In this paper we presented STM, a semantic topic modeling technique designed for source code. STM exploits the basic assumptions of information theory and cluster hypothesis to provide a stable, easy-to-implement, and efficient procedure for detecting and representing linguistic topics in source code.

Ten software systems from different application domains were used to experimentally determine the configuration settings of STM. Our analysis showed that the clustering algorithm Average Linkage, combined with Normalized Google Distance (NGD) as a semantic

distance measure, was able to generate the most coherent clusters of source code terms (topics). These configurations were identified using a semantically aware cluster quality function (8) that was crafted to capture the notion of semantic coherence of generated topics. The popular *word intrusion* and *topic intrusion* tests, which are commonly employed in NLP topic modeling research, were used to evaluate the expressiveness of STM topics. Our experiment was conducted using eight domain experts from three different software systems. The results showed that our study participants found STM topics to be more coherent than topics generated by other classical topic modeling techniques, including LDA and LSA-SC.

A case study was conducted to demonstrate and examine the operation of STM in practical settings. In particular, STM was used to analyze software change in ten releases of the mobile application *Flym*. The results showed that STM was able to detect fine grained software changes that reflected basic software maintenance activities such as feature addition and refactoring.

The main contribution of this paper is two-fold. First, the introduction of an integrated set of theoretically sound techniques and strategies to design a topic modeling approach that has the potential to overcome the inherent limitations of existing topic modeling techniques in the context of source code. Second, the introduction of a set of human driven topic modeling evaluation strategies to help advance topic modeling research in source code analysis. Finally, the work presented in this paper has opened several research directions to be pursued in our future work. These directions can be described as follows:

- Empirical evaluation: We will continue to evaluate STM using more human participants and over different size systems. Our objective is to capture more aspects of what humans consider to be meaningful topics and use these aspects to further tune our approach.
- Applications: The research in this paper is mainly focused on the methodology (i.e. proposing a new topic modeling technique designed for source code analysis). In our future work, STM will be evaluated across several essential software engineering activities in which topic modeling is often applied, such as, traceability link recovery (Asuncion et al. 2010), refactoring (Bavota et al. 2014), debugging (Andrzejewski et al. 2007), and code evolution analysis (Thomas et al. 2010).
- Tool support: The overarching goal of our research is to develop an approach that can be effectively integrated in practice, resulting in tools that developers can use on a daily basis. To achieve our objective, a set of working prototypes that implement our findings in this paper will be developed. These prototypes will include stand alone tools and plugins that can be integrated into modern integrated development environments (e.g., Eclipse and VS.NET). Working prototypes will enable us to conduct long term usability studies to gain a better understanding of more aspects of our approach, including its scalability, usability, and scope of applicability.

Acknowledgments The authors would like to thank our study participants for their time and feedback and the Institutional Review Board (IRB) at LSU for approving our research. This work was supported by the Louisiana Board of Regents Research Competitiveness Subprogram (LA BoR-RCS), contract number: LEQSF(2015-18)-RD-A-07.

References

Abadi A, Nisenson M, Simionovici Y (2008) A traceability technique for specifications. In: International Conference on Program Comprehension, pp. 103–112

Aggarwal C, Zhai C (2012) A survey of text clustering algorithms. In: Mining Text Data, pp. 77–128. Springer
Andrzejewski D, Mulhern A, Liblit B, Zhu X (2007) Statistical debugging using latent topic models. In: European conference on Machine Learning, pp. 6–17
Anquetil N, Fourrier C, Lethbridge T (1999) Experiments with clustering as a software remodularization method. In: Working Conference on Reverse Engineering, pp. 235–255
Anquetil N, Lethbridge T (1998) Assessing the relevance of identifier names in a legacy software system. In: Conference of the Centre for Advanced Studies on Collaborative Research, pp. 4–14
Asuncion H, Asuncion A, Taylor R (2010) Software traceability with topic modeling. In: International Conference on Software Engineering, pp. 95–104
Baldi P, Lopes C, Linstead E, Bajracharya S (2008) A theory of aspects as latent topics. ACM SIGPLAN Not 43(10):543–562
Barua A, Thomas S, Hassan A (2014) What are developers talking about? An analysis of topics and trends in stack overflow. Empir Softw Eng 19(3):619–654
Bavota G, Gethers M, Oliveto R, Poshyvanyk D, De Lucia A (2014) Improving software modularization via automated analysis of latent topics and dependencies. ACM Trans Softw Eng Methodol 23(1):1–33
Bavota G, Oliveto R, Gethers M, Poshyvanyk D, De Lucia A (2014) Methodbook: Recommending move method refactorings via Relational Topic Models. IEEE Trans Softw Eng 40(7):671–694
Bieman J, Kang B (1995) Cohesion and reuse in an object-oriented system. SIGSOFT Software Engineering Notes 20(SI):259–262
Biggers L, Bocovich C, Capshaw R, Eddy B, Etzkorn L, Kraft N (2014) Configuring Latent Dirichlet Allocation based feature location. Empir Softw Eng 19(3):465–500
Binkley D, Heinz D, Lawrie D, Overfelt J (2014) Understanding LDA in source code analysis. In: International Conference on Program Comprehension, pp. 26–36
Blei D, Griffiths T, Jordan M, Tenenbaum J (2003) Hierarchical topic models and the nested Chinese restaurant process. In: Advances in Neural Information Processing Systems
Blei D, Ng A, Jordan M (2003) Latent Dirichlet Allocation. J Mach Learn Res 3:993–1022
Budiu R, Royer C, Pirolli P (2007) Modeling information scent: A comparison of LSA, PMI and GLSA similarity measures on common tests and corpora. In: Large Scale Semantic Access to Content (Text, Image, Video, and Sound), pp. 314–332
Bullinaria J, Levy J (2007) Extracting semantic representations from word co-occurrence statistics: A computational study. Behav Res Methods 39(3):510–526
van Rijsbergen CJ (1979) Information Retrieval. Butterworths
Caprile B, Tonella P (2000) Restructuring program identifier names. In: International Conference on Software Maintenance, pp. 97–107
Chang J (2010) Not-so-latent Dirichlet allocation: Collapsed Gibbs sampling using human judgments. In: NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk, pp. 131–138
Chang J, Boyd-Graber J, Gerrish S, Wang C, Blei D (2009) Reading tea leaves: How humans interpret topic models. Curran Associates, pp 288–296
Chen T, Thomas S, Nagappan M, Hassan A (2012) Explaining software defects using topic models. In: Working Conference on Mining Software Repositories, pp. 189–198
Chidamber S, Kemerer C (1994) A metrics suite for object oriented design. IEEE Trans Softw Eng 20(6):476–493
Church K, Hanks P (1990) Word association norms, mutual information, and lexicography. Comput Linguist 16(1):22–29
Cilibrasi R, Vitanyi P (2007) The google similarity distance. IEEE Trans Knowl Data Eng 19(3):370–383
De Lucia A, Di Penta M, Oliveto R, Panichella A, Panichelle S (2012) Using IR methods for labeling source code artifacts: Is it worthwhile? In: International Conference on Program Comprehension, pp. 193–202
Dean A, Voss D (1999) Design and Analysis of Experiments. Springer
Deerwester S, Dumais S, Furnas G, Landauer T, Harshman R (1990) Indexing by latent semantic analysis. J Am Soc Inf Sci 41(6):391–407
Deißenböck F, Pizka M (2005) Concise and consistent naming. In: International Workshop on Program Comprehension, pp. 97–106
Demmel J, Kahan W (1990) Accurate singular values of bidiagonal matrices. J Sci Stat Comput 11(5):87–912
Gabel M, Zhendong S (2010) A study of the uniqueness of source code. In: ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 147–156
Gethers M, Poshyvanyk D (2010) Using relational topic models to capture coupling among classes in object-oriented software systems. In: International Conference on Software Maintenance, pp. 1–10
Gethers M, Savage T, Di Penta M, Oliveto R, Poshyvanyk D, De Lucia A (2011) Codetopics: which topic am I coding now? In: International Conference on Software Engineering, pp. 1034–1036

Girolami M, Kabán A (2003) On an equivalence between PLSI and LDA. In: International ACM SIGIR Conference on Research and Development in Informaion Retrieval, pp. 433–434
Gracia J, Trillo R, Espinoza M, Mena E (2006) Querying the web: A multiontology disambiguation method. In: International Conference on Web Engineering, pp. 241–248
Grant S, Cordy J (2010) Estimating the optimal number of latent concepts in source code analysis. In: International Working Conference on Source Code Analysis and Manipulation, pp. 65–74
Griffiths T, Steyvers M (2004) Finding scientific topics. In: The National Academy of Sciences, pp. 5228–5235
Haiduc S, Marcus A (2008) On the use of domain terms in source code. In: IEEE International Conference on Program Comprehension, pp. 113–122
Hearst M, Pedersen J (1996) Reexamining the cluster hypothesis: scatter/gather on retrieval results. In: international ACM SIGIR conference on Research and development in information retrieval, pp. 76–84
Hindle A, Barr E, Su Z, Gabel M, Devanbu P (2012) On the naturalness of software. In: International Conference on Software Maintenance, pp. 837–847
Hindle A, Bird C, Zimmermann T, Nagappan N (2015) Do topics make sense to managers and developers Empir Softw Eng 20(2):479–515
Hofmann T (1999) Probabilistic latent semantic indexing. In: International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 50–57
Howard M, Gupta S, Pollock L, Vijay-Shanker K (2013) Automatically mining software-based, semantically-similar words from comment-code mappings. In: Working Conference on Mining Software Repositories, pp. 377–386
Khatiwada S, Kelly M, Mahmoud A (2016) Stac: A tool for static textual analysis of code. In: International Conference on Program Comprehension, pp. 1–3
Koltcov S, Koltsova O, Nikolenko S (2014) Latent Dirichlet Allocation: Stability and applications to studies of user-generated content. In: ACM Conference on Web Science, pp. 161–165
Kuhn A, Ducasse S, Gírba T (2007) Semantic clustering: Identifying topics in source code. Inf Softw Technol 49(3):230–243
Lau J, Newman D, Karimi S, Baldwin T (2010) Best topic word selection for topic labelling. In: International Conference on Computational Linguistics, pp. 605–613
Li M, Chen X, Li X, Ma B (2004) Vitanyi: The similarity metric. IEEE Trans Inf Theory 50(12):3250–3264
Linstead E, Lopes C, Baldi P (2008) An application of Latent Dirichlet Allocation to analyzing software evolution. In: International Conference on Machine Learning and Applications, pp. 813–818
Linstead E, Rigor P, Bajracharya S, Lopes C, Baldi P (2007) Mining concepts from code with probabilistic topic models. In: International Conference on Automated Software Engineering, pp. 461–464
Liu Y, Poshyvanyk D, Ferenc R, Gyimothy T, Chrisochoides N (2009) Modeling class cohesion as mixtures of latent topics. In: International Conference on Software Maintenance, pp. 233–242
Lo D, Nagappan N, Zimmermann T (2015) How practitioners perceive the relevance of software engineering research. In: Joint Meeting on Foundations of Software Engineering, pp. 415–425
Lohar S, Amornborvornwong S, Zisman A, Cleland-Huang J (2013) Improving trace accuracy through data-driven configuration and composition of tracing features. In: Joint Meeting on Foundations of Software Engineering, pp. 378–388
Lukins S, Kraft N, Etzkorn L (2008) Source code retrieval for bug localization using Latent Dirichlet Allocation. In: Working Conference on Reverse Engineering, pp. 155–164
Lukins S, Kraft N, Etzkorn L (2010) Bug localization using Latent Dirichlet Allocation. Inf Softw Technol 52(9):972–990
Mahmoud A, Bradshaw G (2015) Estimating semantic relatedness in source code. ACM Trans Softw Eng Methodol 25(1):1–35
Mahmoud A, Niu N (2015) On the role of semantics in automated requirements tracing. Requir Eng 20(3):281–300
Mancoridis S, Mitchell B, Chen Y, Gansner E (1999) Bunch: A clustering tool for the recovery and maintenance of software system structures. In: International Conference on Software Maintenance, pp. 50–59
Maskeri G, Sarkar S, Heafield K (2008) Mining business topics in source code using Latent Dirichlet Allocation. In: India software engineering conference, pp. 113–120
Meghan K, Revelle, Poshyvanyk D (2009) Using Latent Dirichlet Allocation for automatic categorization of software. In: International Working Conference on Mining Software Repositories, pp. 163–166
Mei Q, Shen X, Zhai C (2007) Automatic labeling of multinomial topic models. In: International Conference on Knowledge Discovery and Data Mining, pp. 490–499
Mihalcea R, Corley C, Strapparava C (2006) Corpus-based and knowledge-based measures of text semantic similarity. In: National Conference on Artificial Intelligence, pp. 775–780

Mimno D, Wallach H, Talley E, Leenders M, McCallum A (2011) Optimizing semantic coherence in topic models. In: The Conference on Empirical Methods in Natural Language Processing, pp. 262–272
Mitchell B, Mancoridis S (2006) On the automatic modularization of software systems using the Bunch tool. IEEE Trans Softw Eng 32(3):193–208
Neuhaus S, Zimmermann T (2010) Security trend analysis with CVE topic models. In: International Symposium on Software Reliability Engineering, pp. 111–120
Newman D, Bonilla E, Buntine W (2011) Improving topic coherence with regularized topic models. In: Neural Information Processing Systems, pp. 496–504
Newman D, Han Lau J, Grieser K, Baldwin T (2010) Automatic evaluation of topic coherence. In: Annual Conference of the North American Chapter of the Association for Computational Linguistics, pp. 100–108
Newman D, Noh Y, Talley E, Karimi S, Baldwin T (2010) Evaluating topic models for digital libraries. In: Annual Joint Conference on Digital Libraries, pp. 215–224
Nguyen A, Nguyen T, Al-Kofahi J, Nguyen H, Nguyen T (2011) A topic-based approach for narrowing the search space of buggy files from a bug report. In: Automated Software Engineering, pp. 263–272
Niu N, Mahmoud A (2012) Enhancing candidate link generation for requirements tracing: The cluster hypothesis revisited. In: IEEE International Requirements Engineering Conference, pp. 81–90
Oliveto R, Gethers M, Poshyvanyk D, De Lucia A (2010) On the equivalence of information retrieval methods for automated traceability link recovery. In: International Conference on Program Comprehension, pp. 68–71
Panichella A, Dit B, Oliveto R, Di Penta M, Poshyvanyk D, De Lucia A (2013) How to effectively use topic models for software engineering tasks? An approach based on genetic algorithms. In: International Conference on Software Engineering, pp. 522–531
Panichella1 A, Dit B, Oliveto R, Di Penta M, Poshyvanyk D, De Lucia A (2016) Parameterizing and assembling IR-based solutions for SE tasks using genetic algorithms. In: International Conference on Software Analysis, Evolution, and Reengineering, pp. 522–531
Porteous I, Newman D, Ihler A, Asuncion A, Smyth P, Welling M (2008) Fast collapsed gibbs sampling for Latent Dirichlet Allocation. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 569–577
Porter F (1997) An algorithm for suffix stripping. Morgan Kaufmann Publishers Inc, pp 313–316
Potapenko A, Vorontsov K (2013) Robust PLSA Performs Better Than LDA. The MIT Press, pp 784–787
Recchia G, Jones M (2009) More data trumps smarter algorithms: Comparing Pointwise Mutual Information with Latent Semantic Analysis. Behav Res Methods 41(3):647–656
Salton G, Wong A, Yang C (1975) A vector space model for automatic indexing. Commun ACM 18(11):613–620
Savage T, Dit B, Gethers M, Poshyvanyk D (2010) TopicXP: Exploring topics in source code using Latent Dirichlet Allocation. In: IEEE International Conference on Software Maintenance, pp. 1–6
Schaeffer S (2007) Graph clustering. Computer Science Review 1(1):27–64
Slonim N, Tishby N (2000) Document clustering using word clusters via the information bottleneck method. In: International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 208–215
Sontag D, Roy D (2011) Complexity of inference in latent dirichlet allocation. In: Shawe-Taylor J., Zemel R. S., Bartlett P. L., Pereira F., Weinberger K. Q. (eds) Advances in Neural Information Processing Systems 24, pp. 1008–1016. Curran Associates, Inc
Sridhara G, Hill E, Pollock L, Vijay-Shanker K (2008) Identifying word relations in software: A comparative study of semantic similarity tools. In: International Conference on Program Comprehension, pp. 123–132
Stevens K, Kegelmeyer P, Andrzejewski D, Buttler D (2012) Exploring topic coherence over many models and many topics. In: Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, pp. 952–961
Steyvers M, Griffiths T (2007) Probabilistic topic models. Psychology Press, pp 427–448
Teh Y, Newman D, Welling M (2007) A collapsed variational bayesian inference algorithm for Latent Dirichlet Allocation. In: Advances in Neural Information Processing Systems 19
Than K, Ho TB (2012) Fully sparse topic models. In: European Conference on Machine Learning and Knowledge Discovery in Databases, pp. 490–505
Thomas S (2011) Mining software repositories using topic models. In: International Conference on Software Engineering, pp. 1138–1139
Thomas S, Adams B, Hassan A, Blostein D (2010) Validating the use of topic models for software evolution. In: IEEE Working Conference on Source Code Analysis and Manipulation, pp. 55–64
Thomas S, Hemmati H, Hassan A, Blostein D (2014) Static test case prioritization using topic models. Empir Softw Eng 19(1):182–212
Tian Y, Lo D, Lawall J (2014) Automated construction of a software-specific word similarity database. In: IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering, pp. 44–53

Turney P (2001) Mining the web for synonyms: PMI-IR versus LSA on TOEFL. In: European Conference on Machine Learning, pp. 491–502
Tzerpos V, Holt R (2000) ACDC: An algorithm for comprehension-driven clustering. In: Working Conference on Reverse Engineering, pp. 258–267
Wallach H, Mimno D, McCallum A (2009) Rethinking LDA: Why priors matter. In: Bengio Y., Schuurmans D., Lafferty J., Williams C., Culotta A. (eds) Advances in Neural Information Processing Systems 22, pp. 1973–1981. Curran Associates, Inc
Wallach H, Murray I, Salakhutdinov R, Mimno D (2009) Evaluation methods for topic models. In: International Conference on Machine Learning, pp. 1105–1112
Yang J, Tan L (2014) Swordnet: Inferring semantically related words from software context. Empirical Software Engingeering 19(6):1856–1886
Zhao Y, Karypis G (2002) Evaluation of hierarchical clustering algorithms for document datasets. In: International Conference on Information and Knowledge Management, pp. 515–524

Anas Mahmoud is an assistant professor of Computer Science and Engineering at Louisiana State University. He obtained his Ph.D. in Computer Science and Engineering in 2014 from Mississippi State University. His main research interests include static code analysis, requirements engineering, program comprehension, code evolution analysis, natural language analysis of software, program refactoring, and information foraging. His work has been published at premier software engineering venues such as TOSEM, EMSE, REJ, ICSE, ICPC, and RE. He is currently directing the Software Engineering and Evolution Laboratory (SEEL) at LSU (http://seel.cse.lsu.edu/).

Gary Bradshaw received his Ph.D. in psychology in 1984 from Carnegie Mellon University. He is currently a professor in the department of Psychology at Mississippi State University. His research interests include Web-based education, decision-making and errors, complex problem solving, psychology of scientific discovery and invention, and empirical software engineering. His work in software engineering has appeared in TOSEM, EMSE, and ICSE.