Empirical Software Engineering manuscript No.
(will be inserted by the editor)

A Survey on the Use of Topic Models when Mining Software Repositories

Tse-Hsun Chen · Stephen W. Thomas · Ahmed E. Hassan

Received: date / Accepted: date

Abstract Researchers in software engineering have attempted to improve software development by mining and analyzing software repositories. Since the majority of the software engineering data is unstructured, researchers have applied Information Retrieval (IR) techniques to help software development. The recent advances of IR, especially statistical topic models, have helped make sense of unstructured data in software repositories even more. However, even though there are hundreds of studies on applying topic models to software repositories, there is no study that shows how the models are used in the software engineering research community, and which software engineering tasks are being supported through topic models. Moreover, since the performance of these topic models is directly related to the model parameters and usage, knowing how researchers use the topic models may also help future studies make optimal use of such models. Thus, we surveyed 167 articles from the software engineering literature that make use of topic models. We find that i) most studies centre around a limited number of software engineering tasks; ii) most studies use only basic topic models; iii) and researchers usually treat topic models as black boxes without fully exploring their underlying assumptions and parameter values. Our paper provides a starting point for new researchers who are interested in using topic models, and may help new researchers and practitioners determine how to best apply topic models to a particular software engineering task.

Keywords Topic modeling · LDA · LSI · Survey

1 Introduction and Motivation

Researchers in software engineering have attempted to improve software development by mining and analyzing software repositories, such as source code changes, email archives, bug databases, and execution logs (Godfrey *et al.*, 2008; Hassan,

Tse-Hsun Chen · Stephen W. Thomas · Ahmed E. Hassan
Software Analysis and Intelligence Lab (SAIL)
School of Computing, Queen's University, Canada
E-mail: {tsehsun, sthomas, ahmed}@cs.queensu.ca

2008). Research shows that interesting and practical results can be obtained from mining these repositories, allowing developers and managers to better understand their systems and ultimately increase the quality of their products in a cost effective manner (Tichy, 2010). Particular success has been experienced with *structured* repositories, such as source code, execution traces, and change logs.

However, automated techniques to understand the *unstructured* textual data in software repositories are still relatively immature (Hassan, 2008), even though 80–85% of the data is unstructured (Blumberg and Atre, 2003; Grimes, 2008). Unstructured data is a current research challenge because the data is often unlabeled, vague, and noisy (Hassan, 2008). For example, the Eclipse bug database contains the following bug report titles:

– "`NPE caused by no spashscreen handler service available`" (#112600)
– "`Provide unittests for link creation constraints`" (#118800)
– "`jaxws unit tests fail in standalone build`" (#300951)

This data is *unlabeled* and *vague* because it contains no explicit links to the source code entity (e.g., package, file, or method) to which it refers, or even to a topic or task from some pre-defined ontology. Instructions such as "link creation constraints," with no additional information or pointers, are ambiguous at best. The data is *noisy* due to misspellings and typographical errors ("spashscreen"), unconventional acronyms ("NPE"), and multiple phrases used for the same concept ("unittests", "unit tests"). The sheer size of a typical unstructured repository (for example, Eclipse has received an average of 115 new bug reports a day for the last 10 years), coupled with its lack of structure, makes manual analysis extremely challenging and in many cases impossible. The end result is that this unstructured data is still waiting to be mined and analyzed.

Despite the above-mentioned challenges, mining unstructured repositories has the potential to benefit software development teams in several ways. For example, linking emails to the source code entities that they discuss could provide developers access to the design decisions made about each code entity. Determining which source code entities are related to a new bug report would significantly reduce the maintenance effort that is required to fix the bug. Automatically creating labels for source code entities would allow developers to more easily browse and understand the code, understand how certain concepts are changing over time, and uncover relationships between entities. All of these tasks would help decrease maintenance costs, increase software quality, and ultimately yield pleased, paying customers.

Advances in the field of Information Retrieval (IR), Machine Learning (ML), and statistical learning, especially the development of *statistical topic models* (Blei and Lafferty, 2009; Blei *et al.*, 2003; Griffiths *et al.*, 2007), have helped make sense of unstructured data in other research communities, including the social sciences (Griffiths *et al.*, 2007; Ramage *et al.*, 2009b) and computer vision (Barnard *et al.*, 2003). Topic models, such Latent Semantic Indexing (LSI) (Deerwester *et al.*, 1990) and latent Dirichlet allocation (LDA) (Blei *et al.*, 2003), are models that automatically discover structure within an unstructured corpus of documents, using the statistical properties of its word frequencies. Topic models can be used to index, search, cluster, summarize, and infer links within the corpus, all tasks that were previously manually performed or not performed at all.

In addition to discovering structure, topic models are hold great promise for several reasons. The models require no training data, which makes them easy to use in practical settings (Blei *et al.*, 2003). The models operate directly on the

raw, unstructured text without expensive data acquisition or preparation costs. (The textual data is often preprocessed, for example by removing common English-language stop words and removing numbers and punctuation, but these steps are fast and simple (Marcus *et al.*, 2004)). Most models, even generative statistical models like LDA, are fast and scalable to millions of documents in real time (Porteous *et al.*, 2008). Some topic models are well equipped to handle both synonymy and polysemy, as explained in Section 2.3. Finally, all topic models can be applied to any text-based software repository, such as the identifier names and comments within source code, bug reports in a bug database, email archives, execution logs, and test cases.

Indeed, researchers are beginning to use topic models to mine software repositories. Recent studies focus on concept mining (e.g., Abebe *et al.*, 2013; Cleary *et al.*, 2008; Grant *et al.*, 2008; Kagdi *et al.*, 2012b; Marcus *et al.*, 2004, 2005; Medini, 2011; Poshyvanyk and Marcus, 2007; Poshyvanyk *et al.*, 2006; Revelle *et al.*, 2010; Van der Spek *et al.*, 2008), constructing source code search engines (e.g., Bajracharya and Lopes, 2010; Grechanik *et al.*, 2010; Tian *et al.*, 2009), recovering traceability links between artifacts (e.g., Ali *et al.*, 2014; Antoniol *et al.*, 2008; Asuncion *et al.*, 2010; Biggers *et al.*, 2014; de Boer and van Vliet, 2008; De Lucia *et al.*, 2004, 2007; Hayes *et al.*, 2006; Jiang *et al.*, 2008; Lohar *et al.*, 2013; Lormans and Van Deursen, 2006; Lormans *et al.*, 2006; Marcus and Maletic, 2003; McMillan *et al.*, 2009), calculating source code metrics (e.g., Bavota *et al.*, 2010; Chen *et al.*, 2012; Gall *et al.*, 2008; Gethers and Poshyvanyk, 2010; Hu and Wong, 2013; Kagdi *et al.*, 2010; Linstead and Baldi, 2009; Liu *et al.*, 2009; Marcus *et al.*, 2008; Ujhazi *et al.*, 2010), and clustering similar documents (e.g., Brickey *et al.*, 2012; Galvis Carreño and Winbladh, 2013; Gorla *et al.*, 2014; Kuhn *et al.*, 2005, 2007, 2008, 2010; Lin *et al.*, 2006; Maletic and Marcus, 2001; Maletic and Valluri, 1999; Raja, 2012). Although there are hundreds of studies on applying topic models to software repositories, there is no study that shows how the models are used in the software engineering research community, and which software engineering tasks are being supported through topic models. Moreover, since the performance of these topic models is directly related to the model parameters and usage, knowing how researchers use the topic models may also help future studies make optimal use of such models.

Prior studies focus on studying the use of topic models on a specific SE task. Thus, in this paper, we survey the software engineering field to determine how topic models have thus far been applied to one or more software repositories. We follow the mapping study approach (Kitchenham *et al.*, 2011; Petersen *et al.*, 2008). Our primary goals are to characterize and quantify:

- which topics models are being used,
- which SE tasks are being supported through topic models,
- how researchers are evaluating their results,
- what preprocessing steps are being performed on the data, and
- what are the typical tools and input parameter values.

Although recent studies have shown promising results (e.g., Borg *et al.* (2014); Dit *et al.* (2013c, 2014)), we performed a detailed analysis of the literature and found several limitations. In particular, we find that most studies to date:

- focus on only a limited number of software engineering tasks;

– use only basic topic models; and
– treat topic models as black boxes without fully exploring their underlying assumptions and parameter values.

We examine a total of 167 articles from the Software Engineering literature that use topic models (see Appendix A for details regarding our article selection process). Section 2 discusses background of the field Mining Software Repositories and topic models. Section 3 collects and presents 37 attributes on each article that help quantify and distinguish it from the others. We use the attributes to present aggregated findings, discuss current research trends, and highlight future research opportunities. Section 4 discusses common uses of topic models on different Software Engineering tasks. Section 5 provides a general guideline on how to avoid common pitfalls when applying topic models to support SE tasks. Section 6 discusses possible future research directions. Finally, Section 7 concludes the paper.

2 Background

In this Section, we first provide background information on the field of Mining Software Repositories. We then highlight the differences between mining structured and unstructured data. Finally, we introduce topic models and their background knowledge.

2.1 Mining Software Repositories

Mining Software Repositories (MSR) is a field of software engineering research, which aims to analyze and understand the data repositories related to software development. The main goal of MSR is to make intelligent use of these software repositories to support the decision-making process of software development (Godfrey *et al.*, 2008; Hassan, 2004, 2008; Hassan and Holt, 2005).

Software development produces several types of repositories during its lifetime, detailed in the following paragraphs. Such repositories are the result of the daily interactions between the stakeholders, as well as the evolutionary changes to various software artifacts, such as source code, test cases, bug reports, requirements documents, and other documentation. These repositories offer a rich, detailed view of the path taken to realize a software system, but they must be transformed from their raw form into something usable (Godfrey *et al.*, 2008; Hassan, 2008; Hassan and Xie, 2010; Tichy, 2010; Zimmermann *et al.*, 2005). A prime example of mining software repositories is *bug prediction*. By mining the characteristics of source code entities (such as size, complexity, number of changes, and number of past bugs), researchers have shown how to accurately predict which entities are likely to have future bugs and therefore deserve additional quality control resources.

2.2 Structured vs. Unstructured Data in Software Repositories

The term "unstructured data" is difficult to define and its usage varies in the literature (Bettenburg and Adams, 2010; Manning *et al.*, 2008). In this paper, we adopt the definition given by Manning *et al.* (2008):

> *"Unstructured data is data which does not have clear, semantically overt, easy-for-a-computer structure. It is the opposite of structured data, the canonical example of which is a relational database, of the sort companies usually use to maintain product inventories and personnel records."*

Unstructured data usually refers to natural language text, since such text has no explicit data model. Most natural language text indeed has latent structure, such as parts-of-speech, named entities, relationships between words, and word sense, that can be inferred by humans or advanced machine learning algorithms. However, in its raw, unparsed form, the text is simply a collection of characters with no structure and no meaning to a data mining algorithm. Examples of unstructured data in software repositories include: bug report titles and descriptions; source code linguistic data (i.e., identifier names, comments, and string literals); requirements documents; descriptions and comments in design documents; mailing lists and chat logs; and source control database commit messages.

Structured data, on the other hand, has a data model and a known form. Examples of structured data in a software repository include: source code parse trees, call graphs, inheritance graphs; execution logs and traces; bug report metadata (e.g., author, severity, date); source control database commit metadata (e.g., author, date, list of changed files); and mailing list and chat log metadata.

2.3 Topic Models

A *topic model* (or *latent topic model* or *statistical topic model*) is a method designed to automatically extract *topics* from a corpus of text documents (Anthes, 2010; Blei and Lafferty, 2009; Steyvers and Griffiths, 2007). Here, a topic is a collection of words that co-occurred frequently in the documents of the corpus. Due to the nature of language usage, the words that constitute a topic are often semantically related.

Topic models were originally developed as a means of automatically indexing, searching, clustering, and structuring large corpora of unstructured and unlabeled documents. Within the topic modeling framework, documents can be represented by the topics within them, and thus the entire corpus can be indexed and organized in terms of this discovered semantic structure.

2.3.1 Common Terminology

Topic models share a common vernacular, which we summarize below. To make the discussion more concrete, we use a running example of a corpus of three simple documents (shown in Figure 1).

term (word or token) w: a string of one or more alphanumeric characters.
In our example, we have a total of 101 terms. For example, *predicting, bug, there, have, bug* and *of* are all terms. Terms might not be unique in a given document.

document d: an ordered set of N terms, $w_1, \ldots, w_N$.
In our example, we have three documents: d_1, d_2, and d_3. d_1 has $N = 34$ terms, d_2 has $N = 35$ terms, and d_3 has $N = 32$ terms.

Predicting the incidence of faults in code has been commonly associated with measuring complexity. In this paper, we propose complexity metrics that are based on the code change process instead of on the code.

(a) Document d_1.

Bug prediction models are often used to help allocate software quality assurance efforts (for example, testing and code reviews). Mende and Koschke have recently proposed bug prediction models that are effort-aware.

(b) Document d_2.

There are numerous studies that examine whether or not cloned code is harmful to software systems. Yet, few of these studies study which characteristics of cloned code in particular lead to software defects (or faults).

(c) Document d_3.

Fig. 1 A sample corpus of three documents.

query q: an ordered set of $|q|$ terms created by the user, $q_1, \ldots, q_{|q|}$.
In our example, a user might query for "defects" (with $|q|$=1 term) or "cloned code" (with $|q|$=2 terms).

corpus C: an unordered set of n documents, $d_1, \ldots, d_n$.
In our example, there is one corpus, which consists of $n = 3$ documents: d_1, d_2, and d_3.

vocabulary V: the unordered set of m unique terms that appear in a corpus.
In our example, the vocabulary consists of $m = 71$ unique terms across all three documents: *code, of, are, that, to, the, software,* ....

term-document matrix A: an $m \times n$ matrix whose i^{th}, j^{th} entry is the weight of term w_i in document d_j (according to some weighting function, such as term-frequency).
In our example, we have

$$A = \begin{array}{l|ccc} & d_1 & d_2 & d_3 \\ \hline \textit{code} & 3 & 1 & 2 \\ \textit{of} & 2 & 0 & 2 \\ \textit{are} & 1 & 2 & 1 \\ \ldots & \ldots & & \end{array}$$

indicating that, for example, the term *code* appears in document d_1 with a weight of 3, and the term *are* appears in document d_2 with a weight of 2.

topic (concept) z: an m-length vector of probabilities over the vocabulary of a corpus.
In our example, we might have a topic

$$z_1 = \begin{array}{cccccccc} \textit{code} & \textit{of} & \textit{are} & \textit{that} & \textit{to} & \textit{the} & \textit{software} & \ldots \\ \hline 0.25 & 0.10 & 0.05 & 0.01 & 0.10 & 0.17 & 0.30 & \ldots \end{array}$$

indicating that, for example, when a term is drawn from topic z_1, there is a 25% chance of drawing the term *code* and a 30% chance of drawing the term *software*. (This example assumes a generative model, such as PLSI or LDA. See Section 2.3.3 for the full definitions.)

topic membership vector θ_d: For document d, a K-length vector of probabilities of the K topics.

In our example, we might have a topic membership vector

$$\theta_{d_1} = \begin{array}{cccccc} z_1 & z_2 & z_3 & z_4 & \dots \\ \hline 0.25 & 0.0 & 0.0 & 0.70 & \dots \end{array}$$

indicating that, for example, when a topic is selected for document d_1, there is a 25% chance of selecting topic z_1 and a 70% chance of selecting topic z_3 .

document-topic matrix θ (also called document-topic matrix D): an n by K matrix whose i^{th}, j^{th} entry is the probability of topic z_j in document d_i. Row i of θ corresponds to θ_{d_i}.

In our example, we might have a document-topic matrix

$$\theta = \begin{array}{c|ccccc} & z_1 & z_2 & z_3 & z_4 & \dots \\ \hline d_1 & 0.25 & 0.0 & 0.0 & 0.70 & \dots \\ d_2 & 0.0 & 0.0 & 0.0 & 1.0 & \dots \\ d_3 & 0.1 & 0.4 & 0.2 & 0.0 & \dots \end{array}$$

indicating that, for example, document d_3 contains topic z_3 with probability 20%.

topic-term matrix ϕ (also called topic-term matrix T): a K by m matrix whose i^{th}, j^{th} entry is the probability of term w_j in topic z_i. Row i of ϕ corresponds to z_i.

In our example, we might have a topic-term matrix:

$$\phi = \begin{array}{c|cccccccc} & \textit{code} & \textit{of} & \textit{are} & \textit{that} & \textit{to} & \textit{the} & \textit{software} & \dots \\ \hline z_1 & 0.25 & 0.10 & 0.05 & 0.01 & 0.10 & 0.17 & 0.30 & \dots \\ z_2 & 0.0 & 0.0 & 0.0 & 0.05 & 0.2 & 0.0 & 0.05 & \dots \\ z_3 & 0.1 & 0.04 & 0.2 & 0.0 & 0.07 & 0.10 & 0.12 & \dots \\ \dots & \dots & & & & & & & \end{array}$$

Some common issues arise with any language model:

synonymy: Two terms w_1 and w_2, $w_1 \neq w_2$, are *synonyms* if they possess similar semantics.

homonymy: A term w is a *homonym* if it has multiple semantics.

We note that the term *semantic* is hard to define and takes on different meanings in different contexts. In the field of information retrieval, often a manually-created oracle is used (e.g., WordNet (Miller, 1995)) to determine the semantics of a term (i.e., relationships with other terms).

2.3.2 The Vector Space Model

While not a topic model itself, the *Vector Space Model* (VSM) is the basis for many advanced IR techniques and topic models. The VSM is a simple algebraic model directly based on the term-document matrix (Salton *et al.*, 1975). In the VSM, a document is represented by its corresponding column vector in A. For example, if a vector for a document d was $[0, 1, 1, 0, 0]$, then according to the VSM, d contains the two terms, namely those with index 2 and 3. Likewise, it is

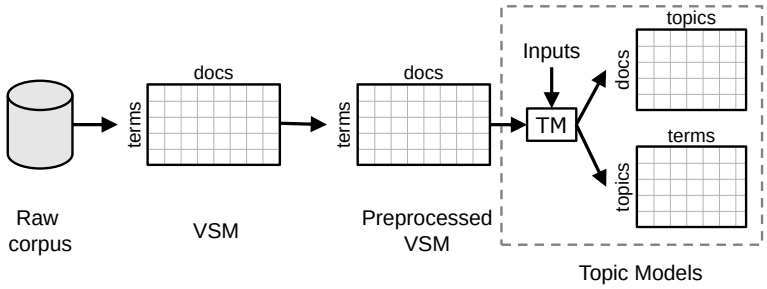


Fig. 2 The path from a raw corpus to a topic model (TM). Here, "Topic Models" includes LSI, ICA, PLSI, LDA, and all LDA variants.

possible to determine which documents contain a given term w by simply selecting the non-zero elements of w's vector in A.

With this representation, one can query the corpus as follows. First, compute the m-lengthed term vector q for the query, as if it were another document in the corpus. Then, compute the semantic-relatedness (often the cosine-similarity) between q and each column of A. Finally, sort the semantic relatedness results to obtain a ranked list of documents that are similar to q. In a similar way, it is possible to determine which of the original documents in the query are most similar to one another.

2.3.3 Topic Models

A *topic model* (or *latent topic model*) is designed to automatically extract *topics* from a corpus of text documents (Anthes, 2010; Blei and Lafferty, 2009; Steyvers and Griffiths, 2007; Zhai, 2008). Here, a topic is a collection of terms that co-occur frequently in the documents of the corpus, for example {*mouse, click, drag, right, left*} and {*user, account, password, authentication*}. Due to the nature of language use, the terms that constitute a topic are often semantically related (Blei *et al.*, 2003).

Topic models were originally developed as a means of automatically indexing, searching, clustering, and structuring large corpora of unstructured and unlabeled documents. Using topic models, documents can be represented by the topics within them, and thus the entire corpus can be indexed and organized in terms of this discovered semantic structure. By representing documents by the lower-dimensional topics, as opposed to terms, topic models (i) uncover *latent* semantic relationships and (ii) allow faster analysis on text (Zhai, 2008). Figure 2 shows the general process of creating basic topic models from a raw corpus; we describe several topic modeling techniques below.

Latent Semantic Indexing Latent Semantic Indexing (LSI) (or *Latent Semantic Analysis* (LSA)) is an information retrieval model that extends the VSM by reducing the dimensionality of the term-document matrix by means of *Singular Value Decomposition* (SVD) (Deerwester *et al.*, 1990). During the dimensionality reduction phase, terms that are related (in terms of co-occurrence) will be

grouped together into topics[1]. This noise-reduction technique has been shown to provide increased performance over VSM in terms of dealing with polysemy and synonymy (Baeza-Yates and Ribeiro-Neto, 1999).

SVD is a factorization of the original term-document matrix A that reduces the dimensionality of A by isolating the *singular values* of A (Salton and McGill, 1983). Since A is likely to be very sparse, SVD is a critical step of the LSI approach. SVD decomposes A into three matrices: $A = TSD^T$, where T is an m by $r = rank(A)$ term-topic matrix, S is the r by r singular value matrix, and D is the n by r document-topic matrix.

LSI augments the reduction step of SVD by choosing a reduction factor, K, which is typically much smaller than the rank of the original term-document matrix r. Instead of reducing the input matrix to r dimensions, LSI reduces the input matrix to K dimensions. There is no perfect choice for K, as it is highly data- and task-dependent. In the literature, typical values range between 50–300. A common approach to choose K for LSI is to examine the amount of variance (e.g., singular values) in the data after computing the SVD (Hu *et al.*, 2003; Jolliffe, 2002). The number of dimensions (K) is chosen such that the variance in the data is lower than a cut-off point, or such that the dimensions that include 70% of the variance are retained (Hu *et al.*, 2003; Jolliffe, 2002).

As in VSM, terms and documents are represented by row and column vectors, respectively, in the term-document matrix. Thus, two terms (or two documents) can be compared by some distance measure between their vectors (e.g., cosine similarity) and queries can by formulated and evaluated against the matrix. However, because of the reduced dimensionality of the term-document matrix after SVD, these measures are more equipped to deal with noise in the data.

Independent Component Analysis *Independent Component Analysis* (ICA) (Comon, 1994) is a statistical technique used to decompose a random variable into statistically independent components (i.e., dimensions). Although not generally considered a topic model, it has been used in similar ways to LSI to model source code documents in a K-dimensional conceptual space.

Like LSI, ICA reduces the dimensionality of the term-document matrix to help reduce noise and associate terms. However, unlike LSI, the resulting dimensions in ICA are statistically independent of one another, which helps capture more variation in the underlying data (Grant and Cordy, 2009).

Probabilistic LSI *Probabilistic Latent Semantic Indexing* (PLSI) (or *Probabilistic Latent Semantic Analysis* (PLSA)) (Hofmann, 1999, 2001) is a generative model that addresses the statistical unsoundness of LSI. Hofmann argues that since LSI uses SVD in its dimension-reduction phase, LSI is implicitly making the unqualified assumption that term counts will follow a Gaussian distribution. Since this assumption is not verified, LSI is "*unsatisfactory and incomplete*" (Hofmann, 1999).

To overcome this assumption, PLSI defines a generative latent-variable model, where the latent variables are topics in documents. At a high level, a generative model has the advantages of being evaluable with standard statistical techniques, such as model checking, cross-validation, and complexity control; LSI could not be

[1] The creators of LSI call these reduced dimensions "concepts", not "topics". However, to be consistent with other topic modeling approaches, we will use the term "topics".

evaluated with any of these techniques. And since the latent variables are topics in documents, PLSI is also well-equipped to more readily handle polysemy and synonymy.

The generative model for each term in the corpus can be summarized with the following steps.

- Select a document d_i with probability $P(d_i)$.
- Select a topic z_k with probability $P(z_k|d_i)$.
- Generate a term w_j with probability $P(w_j|z_k)$.

Given the observations in a dataset (i.e., terms), one can perform inference against this model to uncover the topics $z_1, ..., z_k$. We refer interested readers to the original articles (Hofmann, 1999, 2001).

Subsequent articles (e.g., Blei *et al.*, 2003; Zhai, 2008) show that the generative model of PLSI suffers from at least two critical problems. First, since d is used as an index variable in the first step, the number of parameters that need to be estimated grows linearly with the size of the corpus, which can lead to severe overfitting issues. Second, since the z_k vectors are only estimated for documents in the training set, they cannot be easily applied to new, unseen documents.

Latent Dirichlet Allocation *Latent Dirichlet Allocation* (LDA) is a popular probabilistic topic model (Blei *et al.*, 2003) that has largely replaced PLSI. One of the reasons for its popularity is because it models each document as a multi-membership mixture of K corpus-wide topics, and each topic as a multi-membership mixture of the terms in the corpus vocabulary. This means that there is a set of topics that describe the entire corpus, each document can contain more than one of these topics, and each term in the entire repository can be contained in more than one of these topics. Hence, LDA is able to discover a set of ideas or themes that well describe the entire corpus (Blei and Lafferty, 2009).

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

- Choose a topic vector $\theta_d \sim \text{Dirichlet}(\alpha)$ for document d.
- For each of the N terms w_i:
  - Choose a topic $z_k \sim \text{Multinomial}(\theta_d)$.
  - Choose a term w_i from $\text{p}(w_i|z_k, \beta)$.

Here, $\text{p}(w_i|z_k, \beta)$ is a multinomial probability function, α is a smoothing parameter for document-topic distributions, and β is a smoothing parameter for topic-term distributions.

The two levels of this generative model allow three important properties of LDA to be realized: documents can be associated with multiple topics, the number of parameters to be estimated does not grow with the size of the corpus, and, since the topics are global and not estimated per document, unseen documents can easily be accounted for.

Like any generative model, the task of LDA is that of *inference*: given the terms in the documents, what topics did they come from (and what are the topics)? LDA

Table 1 Example topics from JHotDraw source code version 7.5.1. The labels are automatically-generated based on the most popular bigram in the topic.

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

performs inference with *latent variable models* (or *hidden variable models*), which are machine learning techniques devised for just this purpose: to associate observed variables (here, terms) with latent variables (here, topics). A rich literature exists on latent variable models (Bartholomew, 1987; Bishop, 1998; Loehlin, 1987); for the purposes of this paper, we omit the details necessary for computing the posterior distributions associated with such models. It is sufficient to know that such methods exist and are being actively researched.

For the above-mentioned reasons, it is argued that LDA's generative process gives it a solid footing in statistical rigor—much more so than previous topic models (Blei *et al.*, 2003; Griffiths and Steyvers, 2004; Steyvers and Griffiths, 2007). As such, LDA may be better suited for discovering the latent relationships between documents in a large text corpus.

Table 1 shows example topics discovered by LDA from version 7.5.1 of the source code of JHotDraw (Gamma, 2007), a framework for creating simple drawing applications. For each topic, the table shows an automatically-generated two-word topic label, the top (i.e., highest probable) words for the topic, and the top three matching Java classes in JHotDraw. The topics span a range of concepts, from opening files to drawing Bezier paths. The discovered topics intuitively make sense and the top-matching classes match our expectations—there seems to be a natural match between the "Bezier path" topic and the `CurvedLinear` and `BezierFigure` classes.

A prior study by Wallach *et al.* (2009a) shows that choosing a larger K for LDA does not significantly affect the quality of the generated topics. The additional topics are rarely used (i.e., noise) in the LDA sampling process and may be filtered out. For example, if the data has only 10 topics, running LDA with 15 topics will likely identify 10 real topics and five noise topics (Wallach *et al.*, 2009a). Topic filtering approaches may vary for different domains, and future studies are needed to examine the best approach to filter these noise topics. On the other hand, choosing a small K may be more problematic, since the information (i.e., topics) cannot be separated precisely (Wallach *et al.*, 2009a).

Variations of LDA Several variants and offshoots of LDA have been proposed. All of these variants apply additional constraints on the basic LDA model. Although promising, the software engineering literature usually does not make use of many

of these variants, and therefore we omit a detailed presentation of each. However, the details can be found in references to the original papers.

- *Hierarchical Topic Models* (HLDA) (Blei *et al.*, 2004, 2010). HLDA discovers a tree-like hierarchy of topics within a corpus, where each additional level in the hierarchy is more specific than the previous. For example, a super-topic "user interface" might have sub-topics "toolbar" and "mouse events".
- *Cross-Collection Topic Models* (ccLDA) (Paul, 2009). ccLDA discovers topics from multiple corpora, allowing the topics to exhibit slightly different behavior in each corpus. For example, a "food" topic might contain the words {*food cheese fish chips*} in a British corpus and the words {*food cheese taco burrito*} for a Mexican corpus.
- *Supervised Topic Models* (sLDA) (Blei and McAuliffe, 2008). sLDA considers documents that are already marked with a response variable (e.g., movie reviews with a numeric score between 1 and 5), and provides a means to automatically discover topics that help with the classification (i.e., predicting the response variable) of unseen documents.
- *Labeled LDA* (LLDA) (Flaherty *et al.*, 2005; Ramage *et al.*, 2009a). LLDA takes as input a text corpus in which each document is labeled with one or more labels (such as Wikipedia) and discovers the term-label relations. LLDA discovers a set of topics for each label and allows documents to only display topics from one of its labels.
- *Correlated Topic Models* (CTM) (Blei and Lafferty, 2007). CTM discovers the correlation between topics and uses the correlation when assigning topics to documents. For example, a document about the "genetics" topic is more likely to also contain the "disease" topic than the "X-ray astronomy" topic.
- *Networks Uncovered by Bayesian Inference* (Nubbi) (Chang *et al.*, 2009). Nubbi discovers relationships between pairs of entities in a corpus, where entities are specified as inputs into the model (e.g., people or places). For example, if the entities *George W. Bush* and *Gerald Ford* were input into Nubbi as entities, along with a corpus of political documents, then Nubbi might connect *George W. Bush* to *Gerald Ford* through a "republican" topic.
- *Author-Topic Model* (Rosen-Zvi *et al.*, 2004). The author-topic model considers one or more authors for each document in the corpus. Each author is then associated with a probability distribution over the discovered topics. For example, the author *Stephen King* would have a high probability with the "horror" topic and a low probability with the "dandelions" topic.
- *Polylingual Topic Models* (PLTM) (Mimno *et al.*, 2009). PLTM can handle corpora in several different languages, discovering aligned topics in each language. For example, if PLTM were run on English and German corpora, it might discover the aligned "family" topics {*child parent sibling*} and {*kind eltern geschwister*}.
- *Relational Topic Models* (RTM) (Chang and Blei, 2009). RTM models documents as does LDA, as well as discovers links between each pair of documents. For example, if document 1 contained the "planets" topic, document 2 contained the "asteroids" topic, and document three contained the "Michael Jackson" topic, then RTM would assign a stronger relationship between documents 1 and 2 than between documents 1 and 3 or documents 2 and 3, because topics 1 and 2 are more closely related to each other.

- *Markov Topic Models* (MTM) (Wang *et al.*, 2009). Similar to the Cross-Collection Topic Model, MTM discovers topics from multiple corpora, allowing the topics to exhibit slightly different behavior in each corpus.
- *Pachinko Allocation Model* (PAM) (Li and McCallum, 2006). PAM provides connections between discovered topics in an arbitrary directed acyclic graph. For example, PAM might connect the "language processing" topic to the "speech recognition" topic, but not to the "snorkeling" topic.
- *Topic Modeling with Network Regularization* (TMN) (Mei *et al.*, 2008). TMN can model corpora which have networks defined between documents, such as social networks or call-graph dependencies. TMN discovers topics that overlay on the network in a meaningful way. For example, if a network was formed from author-coauthor relationships in academic articles, then topics that are assigned to author A have a high likelihood of being assigned to one of the coauthors of author A.
- *Biterm Topic Model* (BTM) (Yan *et al.*, 2013). Topic models generally perform poorly on short documents. BTM is designed for short documents (e.g., tweets). BTM learns topics by modeling the word co-occurrence patterns (i.e., biterms). Recent experiments show that BTM generated topics are more prominent, coherent topics, and include more relevant words (according to manual analysis of the topics) (Yan *et al.*, 2013).

2.3.4 Topic Evolution Models

Several advanced techniques have been proposed to extract the *evolution* of a topic in a time-stamped corpus—how the usage of a topic (and sometimes the topic itself) changes over time as the terms in the documents are changed over time. Such a model is usually an extension to a basic topic model that accounts for time in some way. We call such a model a *topic evolution model*.

Initially, the Dynamic Topic Model (Blei and Lafferty, 2006) was proposed. This model represents time as a discrete Markov process, where topics themselves evolve according to a Gaussian distribution. This model thus penalizes abrupt changes between successive time periods, discouraging rapid fluctuation in the topics over time.

The Topics Over Time (TOT) (Wang and McCallum, 2006) model represents time as a continuous beta distribution, effectively removing the penalty on abrupt changes from the Dynamic Topic Model. However, the beta distribution is still rather inflexible in that it assumes that a topic evolution will have only a single rise and fall during the entire corpus history.

The Hall model (Hall *et al.*, 2008) applies LDA to the entire collection of documents at the same time and performs post hoc calculations based on the observed probability of each document in order to map topics to versions. Linstead *et al.* (2008a) and Thomas *et al.* (2010) also used this model on a software system's version history. The main advantage of this model is that no constraints are placed on the evolution of topics, providing the necessary flexibility for describing large changes to a corpus.

The Link model, proposed by Mei and Zhai (2005) and first used on software repositories by Hindle *et al.* (2009), applies LDA to each version of the repository *separately*, followed by a post-processing phase to link topics across versions. Once the topics are linked, the topic evolutions can be computed in the same way as in

the Hall model. The post-processing phase must iteratively link topics found in one version to the topics found in the previous. This process inherently involves the use of similarity thresholds to determine whether two topics are similar enough to be called the same, since LDA is a probabilistic process and it is not guaranteed to find the exact same topics in different versions of a corpus. As a result, at each successive version, some topics are successfully linked while some topics are not, causing past topics to "die" and new topics to be "born". Additionally, it is difficult to allow for gaps in the lifetime of a topic.

2.4 Applying Topic Models to SE Data

Before topic models are applied to SE data, several preprocessing steps are generally taken in an effort to reduce noise and improve the resulting models.

- Characters related to the syntax of the programming language (e.g., "`&&`", "`->`") are removed; programming language keywords (e.g., "`if`", "`while`") are removed.
- Identifier names are split into multiple parts based on common naming conventions, such as camel case (`oneTwo`), underscores (`one_two`), dot seperators (`one.two`), and capitalization changes (`ONETwo`)).
- Common English-language stopwords (e.g., "the", "it", "on") are removed.
- Word stemming is applied to find the root of each word (e.g., "changing" becomes "chang"), typically using the Porter algorithm (Porter, 1980). Other word normalization approaches such as lemmatization, which groups the different inflected forms of a word, may also be used.
- In some cases, the vocabulary of the resulting corpus is pruned by removing words that occur in, for example, over 80% or under 2% of the documents (Madsen *et al.*, 2004).

The main idea behind these steps is to capture the semantics of the developers' intentions, which are thought to be encoded within the identifier names and comments in the source code (Poshyvanyk *et al.*, 2007). The rest of the source code (i.e., special syntax, language keywords, and stopwords) are just noise and will not be beneficial to the results of topic models.

3 Research Trends

In this section, we identify and describe the research trends in the area of mining unstructured repositories using topic models. We define a set of attributes that allow us to characterize each of the surveyed articles. Additionally, we define six facets of related attributes, summarized in Table 2.

First and foremost, we are interested in which *topic model* was primarily used in the study: LSI, LDA or some other model. (Note that if an article evaluates its proposed technique, which uses topic model X, against another technique, which uses topic model Y, we only mark the article as using model X. However, if the main purpose of an article is to compare various topic models, we mark the article with all the considered topic models.) Second, we are interested in the *SE task* that was being performed. We include a range of tasks to allow a fine-grained view of that

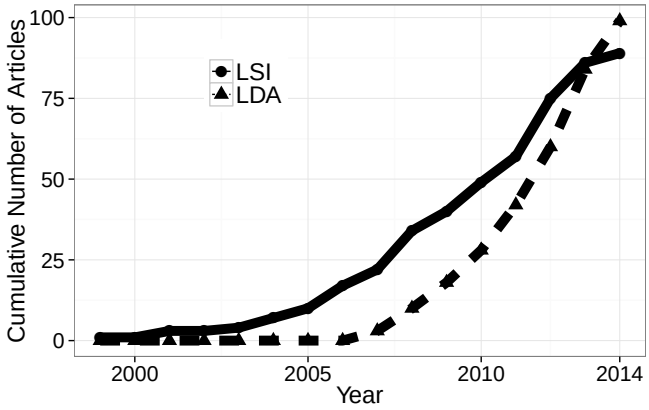


Fig. 3 Trends of LSI and LDA use. The cumulative number of articles indicates the total number of articles published up to the year shown on the x-axis.

literature. Third, we document the *repository* being used in the article. Fourth, we are interested in how the authors of the article *evaluated* their technique, as some topic models are known to be difficult to objectively evaluate. Fifth, we are interested in how the corpus was *preprocessed*, as there are several proposed techniques. Finally, we are interested in which topic modeling *tool* was used in the article, along with which parameter values were chosen for the tool, and how they were chosen.

We manually processed each of the articles in our article set and assigned attribute sets to each. The results allow the articles to be summarized and compared along our six chosen facets.

The results are shown in Appendix B (Tables B1 and B2). Table B1 shows our first four facets: which topic model was used, which software engineering task was being performed, which repository was used, and how the authors evaluated their technique. Table B2 shows our last two facets: which preprocessing steps were taken, and what topic modeling tools and parameters were used. We now analyze the research trends of each facet.

3.1 Facet 1: Which Topic Models Were Used?

The majority of surveyed articles (66%) used LDA or an LDA variant (some papers aim to compare different topic models, so the sum is over 100%), indicating that LDA is indeed a popular choice. On the other hand, 47% of the surveyed used LSI as the primary topic model. As Figure 3 illustrates, the use of LDA is increasing rapidly since its introduction into the software engineering field in 2006. Although we see some articles that compare the performance of different topic models, we only see a few studies (e.g., the study by Dit *et al.* (2013b), Gethers *et al.* (2011c), and Thomas *et al.* (2013)) that combine different topic models. Nevertheless, combining the models may increase the performance significantly (Dietterich, 2000; Thomas *et al.*, 2013).

Table 2 The final set of attributes we collected on each article.

Facet	Attribute	Description
Topic Model	LSI	uses LSI
	LDA	uses standard LDA
	Other	uses ICA, PLSI, or a variant of LDA
SE Task	doc. clustering	performs a clustering of documents
	concept loc.	concept/feature location or aspect-oriented programming
	metrics	derives source code metrics (usually, but not always, for bug prediction)
	trend/evolution	analyzes/predicts source code evolution
	traceability	uncovers traceability links between pairs of artifacts (including bug localization)
	bug predict./debug	predicts bugs/faults/defects in source code, uses statistical debugging techniques, or performs root cause analysis
	org./search coll.	operates on collections of systems (search, organize, analyze)
	other	any other SE task, including bug triaging and clone detection
Repository	source code	uses source code, revision control repository, or software system repository
	email	uses email, chat logs, or forum postings
	req./design	uses requirements or design documents
	logs	uses execution logs or search engine logs
	bug reports	uses bug reports or vulnerability reports
Evaluation	statistical	uses topic modeling statistics, like log likelihood or perplexity
	task specific	uses a task specific method (e.g., classification accuracy)
	manual	performs a manual evaluation
	user study	conducts a user study
Preprocessing	identifiers	includes source code identifiers
	comments	includes source code comments
	string literals	includes string literals in source code
	tokenize	splits camelCase and under_scores
	stem	stems the terms in the corpus
	stop	performs stop word removal
	prune	removes overly common or overly rare terms from vocabulary
Tool Use	tool	name of the used topic model implementation
	K value	for LDA and LSI, the value chosen for the number of topics, K
	K justif.	justification given for the chosen K
	iterations	number of sampling iterations run (if LDA or LDA variant)

Although LSI appeared earlier than LDA, the majority of surveyed articles used LDA or LDA variant. Moreover, we found little research that combines the results of different topic modeling approaches to improve their overall performance.

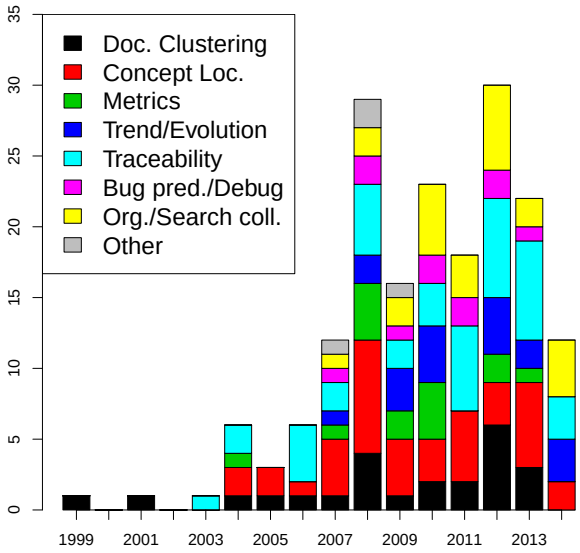


Fig. 4 A stacked bar plot which shows the trends of the investigated tasks. The y-axis shows the number of articles published in a year.

3.2 Facet 2: Which Software Engineering Task Was Supported?

The most popular software engineering tasks in the surveyed articles are traceability link recovery (25% of articles) and concept location (24% of articles). Traceability link recovering is a task well-suited for topic models, since the goal of traceability recovering is to find the textual similarity between pairs of documents. Thus, using the document similarity metrics defined on the topic membership vectors of two documents is a direct implementation of traceability link recovery.

Concept location is an ideal task for topic models, since many researchers (e.g., Baldi *et al.* (2008)) believe that the topics discovered by a topic model are essentially equivalent (or can be directly mapped) to the conceptual concerns in the source code.

The tasks in the "*other*" category include bug triaging (Ahsan *et al.*, 2009; Naguib *et al.*, 2013), search engine usage analysis (Bajracharya and Lopes, 2009, 2010), auditor support for exploring the implementation of requirements (de Boer and van Vliet, 2008), analyzing bug report quality (Alipour *et al.*, 2013; Dit *et al.*, 2008), estimating the number of topics or other topic model parameters in source code (Dit *et al.*, 2013a; Grant and Cordy, 2010; Panichella *et al.*, 2013), clone identification (Grant and Cordy, 2009; Marcus and Maletic, 2001), finding related code on the web (Poshyvanyk and Grechanik, 2009), web service discovery (Wu *et al.*, 2008), source code summarization and labelling (De Lucia *et al.*, 2012, 2014; Eddy *et al.*, 2013; Iacob and Harrison, 2013; Medini *et al.*, 2012), topic interpre-

tation (Hindle *et al.*, 2012c), and refactoring (Bavota *et al.*, 2012). Section 4.8 has a more detailed discussion on the articles in the "*other*" category.

Figure 4 shows a stacked bar plot of the trend of the tasks performed by the surveyed articles across the years. We see the emergence of articles that conduct studies on collections of systems since 2007 (org./search coll). The reason may be the increased popularity of LDA and its variants, which gives researchers the right technique for analyzing trends and patterns on collections of systems. In addition, debug/bug prediction studies also emerged around 2007, and we notice several articles that are published on a yearly basis on this task since 2007.

Most research performs traceability link recovery or concept location.

3.3 Facet 3: Which Repositories Were Mined?

The overwhelming majority (72%) of the articles mine the source code repository, with the second most being requirements or design documents (21%). One possible reason for the popularity of mining source code is that source code is usually the only repository that is (easily) available to researchers who study open source systems. Requirements and design documents are not created as often for open source systems, and if they are, they are rarely accessible to researchers. Email archives are usually only available for large systems, and when they are available, the archives are not usually in a form that is suited for analysis without complicated preprocessing efforts, due to their unstructured nature. Execution logs are most useful for analyzing ultra-large scale systems under heavy load, which is difficult for researchers to simulate. Bug reports, although gaining in popularity, are typically only kept for large, well-organized projects. In addition, we found that most articles only considered a single snapshot of the repository, even when the repositories history was available.

Figure 5 shows a stacked bar plot of the trends of the mined repositories. We find that source code and requirement repositories have been used by researchers since the earlier 2000. Recently, we see more articles that mine bug reports, logs, mobile applications, and developer discussions.

The majority of prior research mines only the source code of the system, and only mines a single snapshot of the repository.

3.4 Facet 4: How Were the Topic Models Evaluated?

The most typical evaluation method used is task-specific (75% of the surveyed articles). For example, many articles use precision and recall, or the number of recovered bugs as the evaluation approach. This result makes sense, because most researchers are using topic models as a tool to support some software engineering task. Hence, the researchers evaluate how well their technique performed at the given task, as opposed to how well the topic model fit the data, as is typically done in the topic model community.

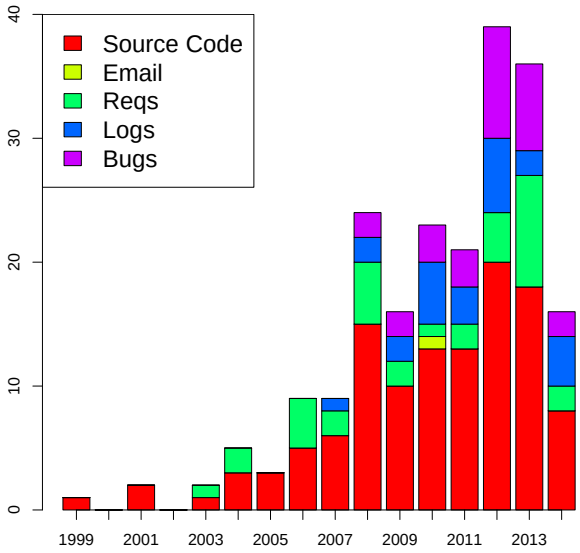


Fig. 5 A stacked bar plot which shows the trends of mined repositories.

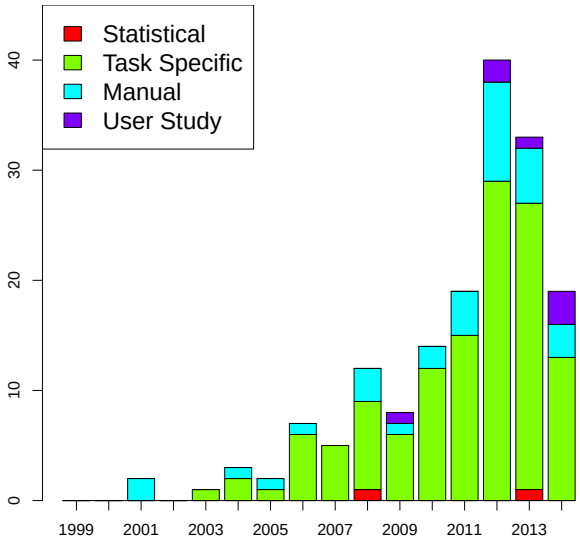


Fig. 6 A stacked bar plot which shows the trends of the task evaluation approaches.

Perhaps surprising is that 19% of articles performed a manual evaluation of the topic model results—an evaluation technique that is difficult and time consuming. Manual evaluation requires humans to manually verify the generated results or topics. This may be due to the seemingly "black-box" nature of topic models—documents are input, results are output, and the rest is unknown. In articles that used topic models such as LDA, manual evaluation is deemed to be the only way to be sure that the discovered topics make sense for software repositories. This may also be due to the limited ground truth available for many software engineering tasks. We also see a few articles that conduct user studies, which usually involve asking developers to verify or interpret the topics/results.

Although articles in the Machine Learning (ML) community, topic models, and natural language processing articles tend to use statistical means to evaluate topic models, only a few surveyed articles used statistical evaluation techniques (e.g., Maskeri *et al.* (2008)). This is perhaps a result of the software engineering community being task-focused, as opposed to model-focused.

Nevertheless, following topic evaluation approaches from the ML community can still benefit MSR studies. ML articles usually evaluate topics using unseen documents (Wallach *et al.*, 2009b). They train the topic model using part of the documents (e.g., 90% of the documents), and estimate the probability of the held-out documents (e.g., rest 10% of the documents). By doing inferences on the held-out documents, one can compute held-out likelihood (e.g., perplexity) for measuring the topic quality (Blei *et al.*, 2003). Blei *et al.* (2003) state that "the perplexity, used by convention in language modeling, is monotonically decreasing in the likelihood of the test data, and is algebraicly equivalent to the inverse of the geometric mean per-word likelihood". In general, perplexity, or similar metrics, measure how well the topic model "generalizes" to the held-out documents. A lower perplexity score indicates better generalization performance.

Wallach *et al.* (2009b) conducted an empirical study on comparing different likelihood computation approaches. They found that Chib-style estimator and left-to-right are better approaches for estimating the quality of topic models. Newman *et al.* (2010) "'search" the topics on Wikipedia, WordNet, and Google search engine, and they use the search result to compute the score for the quality of the topic. They found that using Wikipedia and Google search engine yield the best performance (very close to inter-annotator agreement). However, we note that Wikipedia or WordNet may not be suitable for evaluating topics in the source code, and other code search engine may be used in future studies.

Figure 6 shows a stacked bar plot of the trends of how the tasks are evaluated by the surveyed articles. We see that only a few of the recent articles evaluate the tasks using statistical approach or user studies, whereas most earlier studies make use of task specific evaluation approaches. We also see that, although more articles are published in recent years, most articles still only apply task-specific evaluation approaches.

Most prior research uses task-specific or manual evaluation of the topic models.

3.5 Facet 5: How Was the Data Preprocessed?

Of the articles that analyzed a source code repository, 54% mention that they include the identifiers, 4% mention that they include the comments, and 13% mention that they include string literals. The relatively high percentages for identifiers and comments seem to follow the idea that the semantics of the developers' intention is captured by their choice of identifier names and comments (Abebe *et al.*, 2013; Poshyvanyk *et al.*, 2007).

The majority of articles that analyzed source code created "documents" at the class level (32%), with a close second being the method level (22%). 8% of the articles left the choice of method or class as an input to their tool and reported results on both. 15% of the articles did not specify the level of granularity used.

Studies (Jin *et al.*, 2011; Phan *et al.*, 2008a; Tang *et al.*, 2014) in the Machine Learning and Data Mining community have found that topic models perform poorly on short documents (e.g., methods and logs). As a result, many new approaches have been proposed to handle short documents (Guo and Diab, 2012; Jin *et al.*, 2011; Yan *et al.*, 2013). However, we found there are no MSR studies that discuss such a problem nor ones that adapt one of the more advanced topic models for short documents.

The majority of articles that analyzed source code chose to tokenize terms (55%). One reason for this is that identifier names in source code are often written in a form that lends itself to tokenization (e.g., `camelCase` and `under_score`). By tokenizing these identifier names, the terms are being broken down into their base form and generating a larger likelihood of finding meaningful topics.

To further reduce the size of the vocabulary and increase the effectiveness of topic models, 45% of articles report stemming words, 60% report removing stop words, and 15% report pruning the vocabulary by removing overly- and/or underly-used terms. We did not find any paper that report using lemmatization. Thus, a possible direction may be studying the effect of lemmatization when applying topic models on software engineering data.

In general, many articles were unclear as to how they preprocessed the textual data, even though this step may have a large impact on the results of topic models (Thomas *et al.*, 2013). For example, 15% of the surveyed articles did not mention the document granularity of their technique (e.g., an entire class or an individual method) and 50% of the articles did not indicate whether they used word stemming, a common preprocessing step in the topic model community.

Data preprocessing techniques are not well documented and are not consistent across current research. Even though topic models perform poorly on short documents, many studies still apply topic models on methods and logs etc.

3.6 Facet 6: Which Tools Were Used, and What Parameter Values Were Used?

For LDA-based articles, MALLET (McCallum, 2002) was the most frequently reported tool used (7 times). The other commonly-used tools include GibbsLDA and JGibbLDA (Phan *et al.*, 2008b). For LSI-based articles, no tool was used by more than on article.

Choosing the number of topics. Not reporting the value of K was the norm, with 45% of articles giving no indication of their choice. Of those that did, values ranged between 5 and 1,000, with the most frequent values being between 10 and 500. 61% of the articles that did specify the value of K did not specify *why* that particular value was used. Of the articles that did specify why, 42% came to an optimal value by testing a range of K values and evaluating each in some way (usually task-specific). A single article used an expert's opinion on what the number of topics should be ("*The number of topics for each program are chosen according to domain expert advice.*" (Andrzejewski *et al.*, 2007)), although the actual value was not reported. A few articles also determine K based on the number of unique terms in the source code files (e.g., Xia *et al.* (2013); Xue *et al.* (2012)).

Panichella *et al.* (2013) also proposed an approach to automatically find the optimal LDA parameters for software engineering tasks. They used a genetic algorithm to search for the combinations of LDA parameters that give the best result for traceability, feature location, and software labeling. They found that the choices of the parameters have large impacts on the results. Lohar *et al.* (2013) also used a genetic algorithm to search for the best combination of LSI parameters, as well as preprocessing approaches. Biggers *et al.* (2014) measured the performance of LDA-based feature location approaches when using different configurations. They also proposed a recommendation for LDA-based feature location approaches. Thomas *et al.* (2013) applied different combinations of information retrieval models, model configurations, and preprocessing approaches to find the best classifier for bug localization. They found that after properly preprocessing the source code entities, VSM gives the best performance. Bradford (2008) conducted an empirical study on choosing K for LSI. They found that when K is around 300, LSI provides very good results on moderate-sized corpus. Moreover, checking the amount of variance in the data after applying SVD can be used to find the optimal K for LSI (Wall *et al.*, 2003).

There are a number of studies which proposed approaches to find optimal topic model parameters. Grant and Cordy (2010) tackled the challenge of choosing the optimal number of topics to input into LDA when analyzing source code. The authors' technique varied the number of topics and used a heuristic to determine how well a particular choice is able to identify two pieces of code located in the same document. The authors concluded with general guidelines and case studies.

A prior study by Wallach *et al.* (2009a) in the ML community shows that choosing a larger K for LDA does not significantly affect the quality of the generated topics. The extra topics can be considered noise during the LDA generative process. However, choosing a small K may not separate the information (i.e., information) precisely (Wallach *et al.*, 2009a). In short, researchers can filter out the noise topics in the case of over-fitting K, but under-fitting will make the topics non-separable. Nevertheless, we still see many MSR studies only use a relative small number of topics (Table B1 in Appendix B), and only a few studies that apply topic filtering.

Choosing hyperparameters and sampling iterations. The reporting of other input parameters, such as α, β, and the number of sampling iterations (in the case of LDA) was even more scarcer. 72% of the articles that used LDA did not indicate the number of iterations sampled. Of those that did, values ranged between 50 and 10,000, with 1,000 and 3,000 being the norm. Wallach *et al.* (2009a) in the

ML community found that using optimized hyperparameters result in improved consistency in topic usage. Namely, using optimized hyperparameters, topics won't be dominated by common words (e.g., a topic of pure stopwords), and the topics are more stable when K increases. Hence, important topics and noise topics will be clearly separated. Thus, future studies may consider using hyperparameter optimization for optimizing α and β.

Dit *et al.* (2013c, 2014) found that 95% of their studied articles do not use the same dataset for evaluation, and only 38% of their studied articles compared the proposed approach with prior studies. In our paper, we found that most articles do not report the topic model parameters they use. Thus, it is difficult to reproduce or compare the performance of the approaches proposed by prior studies. We recommend future studies to include a discussion of the parameters used to improve the reproducibility of the research.

Key study design decisions are often not well documented. For instance, 45% of the surveyed articles we studied did not report the value of K (topics or reduction factor) that is used in the topic model, even though it is well known that this choice greatly affects the output of the topic model, and thus the results of the study.

3.7 Conclusions

The above-identified trends reveal many potentials for advancement of the state of the art. First, concept location and traceability linking are the most often addressed software engineering tasks, leaving many tasks under explored.

Second, most research only uses a single topic model, even though research in other communities indicates that combining multiple models can improve overall performance (Misirli *et al.*, 2011).

Finally, research is inconsistent as to which data preprocessing steps are performed, and most articles lack any justification as to why some steps are performed and others are not. In addition, parameter values are often not reported, and when they are, they are not justified or consistent with previous research. Most research rarely explores the sensitivity of topic models to their parameters.

4 Common Uses of Topic Models on Software Engineering Tasks

In this section, we describe and evaluate related work that uses topic models to mine software repositories and perform some software engineering tasks. We organize the work into subsections by software engineering task. We provide a brief description of each task, followed by a presentation of the relevant articles.

4.1 Concept/Feature Location

The task of *concept location* (or *feature location*) is to identify the parts (e.g., documents or methods) of the source code that implement a given feature of the

software system (Rajlich and Wilde, 2002). This is useful for developers wishing to debug or enhance a given feature. For example, if the so-called *file printing* feature contained a bug, then a concept location technique would attempt to automatically find those parts of the source code that implement file printing (i.e., parts of the source code that are executed when the system prints a file).

Related to concept location is *aspect-oriented programming* (AOP), which aims at providing developers with the machinery to easily implement aspects of functionality whose implementation spans over multiple source code documents.

4.1.1 LSI-based Techniques

LSI was first used for the concept location task by Marcus *et al.* (2004), who developed a technique to take a developer query and return a list of related source code documents. The authors showed that LSI provides better results than existing methods (i.e., regular expressions and dependency graphs) and is easily applied to source code, due to the flexibility and light-weight nature of LSI. The authors also noted that since LSI is applied only to the comments and identifiers of the source code, it is language-independent and thus accessible for any system.

Marcus *et al.* (2005) demonstrated that concept location is needed in the case of Object-Oriented (OO) programming languages, contrary to previous beliefs. The authors compared LSI with two other techniques, namely regular expressions and dependency graphs, for locating concepts in OO source code. The authors concluded that all techniques are beneficial and necessary, and each possesses its own strengths and weaknesses.

Poshyvanyk *et al.* (2006) combined LSI and Scenario Based Probabilistic ranking of execution events for the task of feature location in source code. The authors demonstrated that using the two techniques, when applied together, outperform either of the techniques individually.

Poshyvanyk and Marcus (2007) and Poshyvanyk *et al.* (2013) combined LSI and Formal Concept Analysis (FCA) to locate concepts in source code. LSI is first used to map developer queries to source code documents, then FCA is used to organize the results into a concept lattice. The authors found that this technique works well, and that concept lattices are up to four times more effective at grouping relevant information than simple ranking methods.

Cleary *et al.* (2008) compared several IR (e.g., VSM, LSI) and NLP techniques for concept location. After an extensive experiment, the authors found that NLP techniques do not offer much of an improvement over IR techniques, which is contrary to results in other communities.

Van der Spek *et al.* (2008) used LSI to find concepts in source code. The authors considered the effects of various preprocessing steps, such as stemming, stopping, and term weighting. The authors manually evaluated the resulting concepts with the help of domain experts.

Grant *et al.* (2008) used ICA, a conceptually similar model to LSI, to locate concepts in source code. The authors argued that since ICA is able to identify statistically independent signals in text, it can better find independent concepts in source code. The authors showed the viability of ICA to extract concepts through a case study on a small system.

Revelle and Poshyvanyk (2009) used LSI, along with static and dynamic analysis, to tackle the task of feature location. The authors combined the different

techniques in novel ways. For example, textual similarity was used to traverse the static program dependency graphs, and dynamic analysis removed textually-found methods that were not executed in a scenario. The authors found that no technique outperformed all others across all case studies.

Revelle *et al.* (2010) performed data fusion between LSI, dynamic analysis, and web mining algorithms (i.e., HITS and PageRank) to tackle the task of feature location. The authors found that combining all three techniques significantly outperforms any of the individual methods, and outperforms the state-of-the-art in feature location.

Binkley *et al.* (2012) studied possible ways to improve the performance of feature location techniques. They proposed a way to normalize the vocabulary in the source code, which goes beyond simple identifier name splitting. They found that expanding the identifier names (e.g., expanding acronyms) can improve the accuracy of feature location techniques.

4.1.2 LDA-based Techniques

Linstead *et al.* (2007b) were the first to use LDA to locate concepts in source code in the form of LDA topics. Their technique can be applied to individual systems or large collections of systems to extract the concepts found within the identifiers and comments in the source code. The authors demonstrated how to group related source code documents based on comparing the documents' topics.

Linstead *et al.* (2007a) applied a variant of LDA, the Author-Topic model (Rosen-Zvi *et al.*, 2004), to source code to extract the relationship between developers (authors) and source code topics. Their technique allows the automated summarization of "who has worked on what", and the authors provided a brief qualitative argument as to the effectiveness of this technique.

Maskeri *et al.* (2008) applied LDA to source code to extract the business concepts embedded in comments and identifier names. The authors applied a weighting scheme for each keyword in the system, based on where the keyword is found (e.g., class name, parameter name, method name). The authors found that their LDA-based technique is able to successfully extract business topics, implementation topics, and cross-cutting topics from source code.

Baldi *et al.* (2008) proposed a theory that software concerns are equivalent to the latent topics found by statistical topic models. Further, they proposed that aspects are those latent topics that have a high scattering metric. The authors applied their technique to a large set of open-source systems to identify the global set of topics, as well as perform a more detailed analysis of a few specific systems. The authors found that latent topics with high scattering metrics are indeed those that are typically classified as aspects in the AOP community.

Savage *et al.* (2010) introduced a topic visualization tool, called Topic$_{XP}$, which supports interactive exploration of discovered topics located in source code.

Grant *et al.* (2011b) developed a technique for visualizing the software maintenance activities captured by LDA. They also examined the relationship between the maintenance activities and concept location captured by LDA.

Nie and Zhang (2012) computed the topic cohesion and coupling using software dependency network to improve the accuracy of LDA-based feature location techniques. They found that their approach can improve the effectiveness of feature location techniques.

Bassett and Kraft (2013) proposed a new weighting algorithm for the vocabularies in the source code. They evaluated the effect of their weighting algorithm on LDA-based concept location technique, and found that they could achieve a statistically significant improvement to the accuracy of concept location techniques.

4.2 Traceability Recovery and Bug Localization

An often-asked question during software development is: "*Which source code document(s) implement requirement X?*" *Traceability recovery* aims to automatically uncover links between pairs of software artifacts, such as source code documents and requirements documents. This allows a project stakeholder to trace a requirement to its implementation, for example to ensure that it has been implemented correctly (or at all!). Traceability recovery between pairs of source code documents is also important for developers wishing to learn which source code documents are somehow related to the current source code file being worked on. *Bug localization* is a special case of traceability recovery in which traceability links between bug reports and source code are sought.

4.2.1 LSI-based Techniques

Marcus and Maletic (2003) were the first to use LSI to recover traceability links between source code and documentation (e.g., requirements documents). The authors applied LSI to the source code identifiers and comments and the documentation, then computed similarity scores between each pair of documents. A user could then specify a similarity threshold to determine the actual links. The authors compared their work to a VSM-based recovery technique and found that LSI performs at least as good as VSM in all case studies.

De Lucia *et al.* (2004) integrated a traceability recovery tool, based on LSI, into a software artifact management system called ADAMS. The authors presented several case studies that use their LSI-based technique to recover links between source code, test cases, and requirements documents. In subsequent work, De Lucia *et al.* (2006) proposed an incremental technique for recovering traceability links. In this technique, a user semi-automatically interacts with the system to find an optimal similarity threshold between documents (i.e., a threshold that properly discriminates between related and unrelated documents). The authors claimed that a better threshold results in fewer links for the developer to consider, and thus fewer chances for error, making human interaction a necessity.

Hayes *et al.* (2006) evaluated various IR techniques for generating traceability links between various high- and low-level requirements, concentrating on the tf-idf and LSI models. The authors implemented a tool called RETRO to aid a requirements analyst in this task. The authors concluded that, while not perfect, IR techniques provide value to the analyst.

Lormans and Van Deursen (2006) evaluated different linking strategies (i.e., thresholding techniques) for traceability recovering using LSI by performing three case studies. The authors concluded that LSI is a promising technique for recovering links between source code and requirements documents and that different linking strategies result in different results. However, the authors observed that no linking strategy is optimal under all scenarios. In subsequent work, Lormans

(2007) introduced a framework for managing evolving requirements (and their traceability links) in a software development cycle. Their technique uses LSI to suggest candidate links between artifacts.

Lormans *et al.* (2006) used LSI for constructing *requirement views*, which are different views of requirements. For example, one requirement view might display only requirements that have been implemented. The authors implemented their tool, called ReqAnalyst, and used it on several real-world case studies.

De Lucia *et al.* (2007) were the first to perform a human case study, which evaluated the effectiveness of using LSI for recovering traceability links during the software development process. The authors concluded that LSI is certainly a helpful step for developers, but that its main drawback is the inevitable trade off between precision and recall.

Jiang *et al.* (2008) proposed an incremental technique to maintaining traceability links as a software system evolves over time. The authors' technique, called incremental LSI, uses links (and the LSI matrix) from previous versions when computing links for the current version, thus saving computation effort.

de Boer and van Vliet (2008) developed a tool to support auditors in locating documentation of interest. The tool, based on LSI, suggests to the auditor documents that are related to a given query, as well as documents that are semantically related to a given document. Such a process gives the auditor, who is unfamiliar with the documentation, a guide to make it easier to explore and understand the documentation of a system.

Antoniol *et al.* (2008) introduced a tool called Reuse or Rewrite (ReORe) to help stakeholders decide if they should update existing code (for example, to introduce new functionalities) or completely rewrite from scratch. ReORe achieves this by using a combination of static (LSI), manual, and dynamic analysis to create traceability links between existing requirements and source code. The stakeholders can then review the recovered traceability links to decide how well the current system implements the requirements.

McMillan *et al.* (2009) used both textual (via LSI) and structural (via Evolving Inter-operation Graphs) information to recover traceability links between source code and requirements documents. The authors performed a case study on a small but well-understood system, `CoffeeMaker`. The authors demonstrated that combining textual and structural information modestly improves traceability results in most cases.

Islam *et al.* (2012b) used LSI to link test cases and software requirements. They prioritize test cases using the test execution time and the requirements to which a test is linked. They showed that their approach outperformed the baseline techniques and provided additional improvements.

4.2.2 Comparison Studies

While the majority of researchers only evaluate their technique with respect to a single topic model, a few have directly compared the performance of multiple topic models.

Lukins *et al.* (2008, 2010) used LDA for bug localization. The authors first build an LDA model on the source code at the method level, using the standard preprocessing steps. Then, given a bug report, the authors compute the similarity of the text content of the bug report to all source code documents. They then return

the top ranked source code documents. By performing case studies on Eclipse and Mozilla (on a total of 3 and 5 bug reports, respectively), the authors find that LDA often outperforms LSI. We note that the authors use manual query expansion, which may influence their results.

Nguyen *et al.* (2011a) introduced a new topic model based on LDA, called BugScout, in an effort to improve the performance of bug localization techniques. BugScout explicitly considers past bug reports, in addition to identifiers and comments, when representing source code documents, using the two data sources together to identify key technical concepts. The authors applied BugScout to four different systems and found that BugScout improves performance by up to 20% over LDA applied only to source code.

Rao and Kak (2011) compared several IR models for bug localization, including VSM, LSI, and LDA, as well as various combinations. The authors performed a case study on a small dataset, iBUGS (Dallmeier and Zimmermann, 2007), and concluded that simpler IR models often outperform more sophisticated models.

Capobianco *et al.* (2009) compared the ability of four different techniques (Vector Space Model, LSI, Jenson-Shannon, and B-Spline) to recover traceability links between source code, test cases, and UML diagrams. The authors found that the B-Spline method outperforms VSM and LSI, and is comparable to the Jenson-Shannon method.

Oliveto *et al.* (2010) compared the effectiveness of four IR techniques for traceability recovery: Jenson-Shannon, VSM, LSI, and LDA. The authors showed that LDA provides unique dimensionality compared to the other four techniques.

Asuncion *et al.* (2010) introduced a tool called TRASE that uses LDA for prospectively, as opposed to retrospectively, recovering traceability links amongst diverse artifacts in software repositories. This means that developers can create and maintain traceability links as they work on the system. The authors demonstrated that LDA outperforms LSI in terms of precision and recall.

Beard *et al.* (2011) compared the performance of LSI and LDA for bug localization, after including structural information (e.g., call graph). They found that, after including structural information, both topic models had similar performance.

4.3 Source Code Metrics

Bug prediction (or *defect prediction* or *fault prediction*) tries to automatically predict which parts (e.g., classes or methods) of the source code are likely to contain bugs. This task is often accomplished by collecting metrics on the source code, training a statistical model to the metrics of documents that have known bugs, and using the trained model to predict whether new documents will contain bugs.

Often, the state of the art in bug prediction is advanced either by the introduction of new metrics or by the use of a previously unexplored statistical model (e.g., Kamei *et al.* (2010), Nguyen *et al.* (2010), Shihab *et al.* (2010)). An entire suite of metrics have thus far been introduced, counting somewhere in the hundreds. Additionally, dozens or hundreds of statistical models have been applied with varying degrees of success.

The majority of metrics are measured directly on the code (e.g., code complexity, number of methods per class) or on the code change process (methods that are frequently changed together, number of methods per change). However,

researchers have used topic models to introduce *semantic* or *conceptual* metrics, which are mostly based on the comments and keywords in the source code.

4.3.1 LSI-based Metrics

Marcus *et al.* (2008) introduced a new class cohesion metric, called the Conceptual Cohesion of Classes (C3), for measuring the cohesion of a program entity. The metric is based on the semantic information in the class, such as identifier names and comments, and is computed using LSI. Highly cohesive entities are thought to follow better design principles and are shown to correlate negatively with program faults. Bavota *et al.* (2010) used the C3 metric in developing a technique to support the automatic refactoring of so-called blob classes (i.e., classes that contain too much functionality and thus have a low cohesion score). Kagdi *et al.* (2010) used a similar metric, the conceptual similarity between pairs of source code methods, as a part of a novel change impact analysis technique.

Gall *et al.* (2008) extensively evaluated a suite of semantic metrics that are computed on the design and requirements documents and on the source code of a system throughout the development process. Some of the metrics are based on LSI. Through three case studies, the authors found significant correlation between metrics measured on design and requirements documents and the same metrics measured source code, providing strong evidence of the semantic similarity of these documents. The authors argued that tracking such metrics can help in the detection of problematic or suspect design decisions early in the software development process.

Ujhazi *et al.* (2010) defined two new conceptual metrics that measure the coupling and cohesion of methods in software systems. Both metrics are based on a method's representation in an LSI subspace. The authors compared their new metrics to an existing suite of metrics (including those of Marcus *et al.* (2008)) and found that the new metrics provide statistically significant improvements compared to previous metrics.

4.3.2 LDA-based Metrics

Linstead and Baldi (2009) applied LDA to the bug reports in the GNOME system with the goal of measuring the coherence of a bug report, i.e., how easy to read and how focused a bug report is. This coherence metric is defined as the tangling of LDA topics within the report, i.e., how many topics are found in the report (fewer is better).

Liu *et al.* (2009) applied LDA to source code methods in order to compute a novel class cohesion metric called Maximum Weighted Entropy (MWE). MWE is computed based on the occupancy and weight of a topic in the methods of a class. The authors demonstrated that this metric captures novel variation in models that predict software faults.

Gethers and Poshyvanyk (2010) introduced a new coupling metric, the Relational Topic-based Coupling (RTC) metric, based on a variant of LDA called Relational Topic Models (RTM). RTM extends LDA by explicitly modeling links between documents in the corpus. RTC uses these links to define the coupling between two documents in the corpus. The authors demonstrated that their proposed metric provides value because it is statistically different from existing metrics.

Nguyen *et al.* (2011b) and Chen *et al.* (2012) proposed a number of metrics based on LDA topics to study software defects. Nguyen *et al.* (2011b) predict defects using topic membership values. On the other hand, Chen *et al.* (2012) found that topics can give extra information in statistical models (i.e., logistic regression model) when studying defects. They also found that the more topics a file has, the more likely it will be defect-prone.

Hu and Wong (2013) used a variant of LDA, called citation influence model, to quantify the dependency strength among software components and developers. They used the strength of the dependency and social network properties to predict software defects.

4.4 Statistical Debugging and Root Cause Analysis

Andrzejewski *et al.* (2007) performed statistical debugging with the use of Delta LDA, a variant of LDA. *Statistical debugging* is the task of identifying a problematic piece of code, given a log of the execution of the code. Delta LDA is able to model two types of topics: usage topics and bug topics. Bug topics are those topics that are only found in the logs of failed executions. Hence, the authors were able to identify the pieces of code that likely caused the bugs.

Bose and Suresh (2008) used LSI as a tool for root cause analysis (RCA), i.e., identifying the root cause of a software failure. The authors built and executed a set of test scenarios that exercised the system's methods in various sequences. Then, the authors used LSI to build a method-to-test co-occurrence matrix, which clustered tests that execute similar functionalities, helping to characterize the different manifestations of a fault.

Zawawy *et al.* (2010) presented a framework for reducing the size and complexity of execution logs so that the manual work performed by a log analyst is reduced during RCA. The reduction is accomplished by filtering the log by performing SQL queries and LSI queries. The authors demonstrated that LSI leads to fewer false positives and higher recall during the filtering process.

4.5 Software Evolution and Trend Analysis

Analyzing and characterizing how a software system changes over time, or the *software evolution* (Lehman, 1980) of a system, has been of interest to researchers for many years. Both *how* a software system changes (e.g., it grows rapidly every twelfth month) and *why* a software system changes (e.g., a bug fix) can help yield insights into the processes used by a specific software system as well as software development as a whole.

Linstead *et al.* (2008a) applied LDA to several versions of the source code of a system in an effort to identify the trends in the topics over time. Trends in source code histories can be measured by changes in the probability of seeing a topic at specific version. When documents pertaining to a particular topic are first added to the system, for example, the topics will experience a spike in overall probability.

In a similar effort, Thomas *et al.* (2010) evaluated the effectiveness of topic evolution models for detecting trends in the software development process. The authors applied LDA to a series of versions of the source code and calculated

the popularity of a topic over time. The authors manually verified that spikes or drops in a topic's popularity indeed coincided with developer activity mentioned in the release notes and other system documentation, providing evidence that topic evolution models provide a good summary of the software history.

Hindle *et al.* (2009, 2010) applied LDA to commit log messages in order to see what topics are being worked on by developers at any given time. The authors applied LDA to the commit logs in a 30 day period, and then linked successive periods together using a topic similarity score (i.e., two topics are linked if they share 8 out of their top 10 terms). The authors found LDA to be useful in identifying developer activity trends.

Neuhaus and Zimmermann (2010) used LDA to analyze the Common Vulnerabilities and Exposures (CVE) database, which archives vulnerability reports from many different sources. The authors' goal was to find the trends of each vulnerability, in order to see which are increasing and which are decreasing. The authors found that their results are mostly comparable to an earlier manual study on the same dataset.

Han *et al.* (2012) used LDA to analyze bug reports for HTC and Mozilla overtime. They studied how the topics evolve, and used the topics mined from bug reports to study Android fragmentation and vendor-specific bugs.

Barua *et al.* (2012) applied LDA on Stack Overflow posts for studying the topic trends in programming question and answer websites. The authors found that topics related to mobile application and web development are getting more popular over time. Linares-Vásquez *et al.* (2013) used LDA to study popular topics related to mobile-development on Stack Overflow. Allamanis and Sutton (2013) used LDA on Stack Overflow to study which programming concepts are more confusion. Bajaj *et al.* (2014) used LDA on Stack Overflow questions to study common challenges and misconcepts about web development.

4.6 Document Clustering

Document clustering is the task of grouping related documents together, usually to enhance program understanding or reduce a developer's searching effort (Kuhn *et al.*, 2005, 2007). Documents can be clustered using any of several possible attributes, including their semantic similarity or dependency graphs.

Maletic and Marcus (2001); Maletic and Valluri (1999) first applied LSI to cluster source code documents. The authors claimed that such a clustering can improve program comprehension during the maintenance and evolutionary phases of the software development cycle. The authors found that LSI produces useful clusters and, since LSI is automated, can be of significant value to developers.

In a similar effort, Kuhn *et al.* (2005, 2007) introduced a tool named HAPAX for clustering source code documents. The authors extended the work by Maletic and Marcus (2001) by visualizing the resulting clusters and providing each cluster with a name based on all the words in the class, not just the class names.

Lin *et al.* (2006) introduced a tool called Prophecy that allows developers to search the Java API for groups of related functionalities. The authors applied LSI to the Javadocs of the Java API to find similarities in their functionalities. A developer can then search the LSI index to yield a cluster of related classes.

Kuhn *et al.* (2008, 2010) built a two dimensional map of a software system, where the positions of entities and distances between entities are based on their vocabularies. LSI is used to reduce the dimensionality of the document-term matrix so that similar documents can be closely aligned on the map. This *software cartography* can help developers understand the layout and relationships of their source code.

4.7 Organizing and Searching Software Repositories

Kawaguchi *et al.* (2006) presented a tool called MUDABlue for automatically organizing large collections of open-source software systems (e.g., SourceForge and Google Code) into related groups, called software categories. MUDABlue applies LSI to the identifier names found in each software system. The authors demonstrated that MUDABlue can achieve recall and precision scores above .80, compared with manually created tags of the systems.

Tian *et al.* (2009) developed LACT, a technique to categorize systems based on their underlying topics. This work is similar in nature to Kawaguchi *et al.* (2006), except this work employs LDA instead of LSI. The authors compared their technique to MUDABlue and concluded that the techniques are comparable in effectiveness.

Linstead *et al.* (2008b,c) introduced and used an Internet-scale repository crawler, Sourcerer, to analyze a large set of software systems. The authors applied LDA and the Author-Topic model to extract the concepts in source code and the developer contributions in source code, respectively. The authors also defined new techniques for searching for code, based on the extracted topic model. Sourcerer can be used to analyze existing systems (i.e., view most popular identifier names and LDA topics) as well as search for modules which contain desired functionality.

Poshyvanyk and Grechanik (2009) proposed a technique called S^3 for searching, selecting, and synthesizing existing systems. The technique is intended for developers wishing to find code snippets from an online repository matching their current development needs. The technique builds a dictionary of available API calls and related keywords, based on online documentation. Then, developers can search this dictionary to find related code snippets. LSI is used in conjunction with Apache Lucene to provide the search capability.

Although not using topic models directly, Haiduc *et al.* (2013) found that reforming and optimizing search queries can significantly improve code search performance.

4.8 Other Tasks

Clone Detection. Marcus and Maletic (2001) were the first to detect high-level clones (Bellon *et al.*, 2007; Rahman *et al.*, 2012; Roy *et al.*, 2009) of source code methods by computing the semantic similarity between pairs of methods. The authors used LSI to cluster related methods together in *concept space* (i.e., a K-dimensional representation of a document, based on the document's topic memberships), and tight clusters represents code clones. Despite low levels of precision,

the authors argued that this technique is cheap and can therefore be used in conjunction with existing clone detection techniques to enhance the overall results.

Grant and Cordy (2009) used ICA to detect method clones. The authors argued that since ICA can identify more distinct signals (i.e., topics) than LSI, then the conceptual space used to analyze the closeness of two methods will be of higher effectiveness. The authors performed a small case study on the Linux `kernel` package, but do not compare their results to LSI.

Bug Triaging. Ahsan *et al.* (2009) aimed to create an automatic bug triaging system, which determines which developer should address a given bug report. The authors extracted the textual content from the titles and summaries of a system's bug reports and applied LSI to obtain a reduced term-document matrix. Then, various classifiers mapped each bug report to a developer, trained on previous bug reports and related developers. In the best case, this technique achieved 45% classification accuracy.

Yang *et al.* (2014) map bug reports to topics, then use such mapping to recommend developers that can work on a newly filed bug report.

The number of bug reports and bug report categories may vary significantly, which may affect the result of automated bug report classification using Machine Learning algorithms. Thus, Somasundaram and Murphy (2012) combine both LDA and other Machine Learning algorithms to generate consistent bug report classification results. They found that combining LDA and the Kullback Leibler divergence algorithm yields the most consistent results across all components with various sizes.

Measuring Cohesion of the Comments in Bug Reports. Dit *et al.* (2008) measured the cohesion of the content of a bug report by applying LSI to the entire set of bug reports and then calculating a similarity measure on each comment within a single bug report. The authors compared their metrics to human-generated analysis of the comments and found a high similarity.

Search Query Analysis. Bajracharya and Lopes (2009, 2010) applied LDA to a usage log of a popular code search engine (Koders) to analyze the user queries over time. Their goal was to determine which topics are the most popular search topics, and whether the search engine provides users with the features that they need to identify the code they want. They found LDA to be an effective tool for such a task.

Software Verification. Thomas *et al.* (2014) applied LDA to prioritize test cases based on the semantic differences in the test cases. They found that their topic model-based approach outperforms traditional black-box test case prioritization approaches. Islam *et al.* (2012b) also proposed approaches to prioritize test cases by using links between test requirement and source code. Gorla *et al.* (2014) applied LDA on the description of mobile applications, and cluster the applications based on the LDA generated topics. Then, they detect mobile applications that have anomalous application permissions within the cluster, and such applications are possibly malicious. Chen *et al.* (2015) apply LDA to find and predict the topics that are less-tested and defect-prone. By focusing the testing resources on the less-tested and defect-prone topics, the maintenance effort can be allocated effectively.

Web-service Discovery. Wu *et al.* (2008) tackled the challenge of building a semantic-based Web-service discovery tool. Their technique, built on LSI, allows the automatic discovery of Web services based on concepts, rather than keywords.

5 A Guideline for Performing Different Software Engineering Tasks Using Topic Models

In this section, we organize the results of our mapping study of the surveyed articles, and present a discussion on the common pitfalls when applying topic models on SE tasks.

5.1 How the Surveyed Articles Use Topic Models for Different Tasks

For each task, we look at three dimensions: 1) the topic model that is used; 2) the repositories that are mined; and 3) how the task is evaluated. In other words, given a software engineering task, we want to answer which topic model is usually used, which repositories are often mined, and how do prior studies evaluate the task. The results may help new researchers and practitioners determine how to best apply topic models to a particular software engineering task.

Table 3 shows how the surveyed articles support each of the software engineering tasks. We focus on the eight tasks that we previously identified (Table 2), and we show the percentage of the surveyed articles that uses each kind of topic model, repository, and evaluation approach. We find that LSI is usually used for document clustering and traceability. Recent studies (e.g., Rao and Kak (2011); Thomas *et al.* (2013)) show that LSI outperforms LDA for traceability, which may explain why most studies choose LSI over LDA for this particular task. We also find that LDA is more often used for trend analysis and for analyzing a collection of systems. The reason may be that LDA can generate human-readable topics, so researchers can interpret and identify the trends of each topic more easily.

We find that most tasks only analyze source code, and rarely use other repositories. One exception is the traceability task, since doing traceability requires two kinds of documents (e.g., source code and requirements). We also find that prior studies evaluate each task differently, and some tasks often lack any evaluation. For example, an evaluation approach is usually missing for studies that analyze collections of systems (around 50% of such studies have an evaluation). We also find that user study is only performed for trend analysis and traceability; whereas other studies do not perform any user studies. Thus, future studies on using topic models for trend analysis and traceability may consider performing user studies for their evaluation.

5.2 Common Pitfalls when Applying Topic Models on Software Engineering Tasks

In Section 4, we discussed how topic models are commonly used in different SE tasks. In this subsection, we further discuss the common pitfalls when applying topics models on Software Engineering tasks.

5.2.1 Choosing Parameters

Although researchers in SE have proposed several approaches to determine the best topic model parameters, we see very few studies that adapt the approaches

proposed by the Machine Learning community. For example, Hierarchical Topic Models (HTM) may be used for estimating the number of topics in a system (Blei *et al.*, 2004, 2010). Hence, HTM may be used to find or verify the optimal number of topics (K) for the examined SE data. In addition, the topic evaluation approaches that are used in the Machine Learning community (e.g., perplexity) can also be used to find the best topic parameters. Future studies may consider using the approach proposed by the Machine Learning community to choose the topic model parameters.

Current studies usually use topic models as a black box, and do not consider the effect of different parameter on the SE task. As a result, future studies may want to examine the effect of different topic model parameters on SE tasks. For example, there is no clear guideline on how varying the LDA hyperparameters may affect the result of feature location. Providing such guideline can help MSR studies choose better topic model parameters.

5.2.2 Labelling and Interpreting Topics

Giving meaningful labels to topics is important for program comprehension. However, interpreting topics is difficult and may be subjective. Researchers in SE have proposed several approaches for labelling the topics. De Lucia *et al.* (2012, 2014) applied topic models on labelling source code artifacts, and compared the automatically generated labels with the human annotated labels. They found that topic models have higher advantages when used on labelling source code artifacts with higher verbosity, or when the artifacts require much effort for humans to label. Medini *et al.* (2012) applied LSI to generate labels for execution traces. The authors compare the generated labels with manually annotated labels, and they found that the generated are informative and useful. Researchers in the ML community proposed an approach to automatically label topics in a more subjective fashion by minimizing the Kullback-Leibler divergence between word distributions (Mei *et al.*, 2007). Their case study shows that their approach can generate more meaningful and useful topic labels.

Although automatic topic labelling can be helpful, recent studies show that interpreting topics may not always be an easy task. Hindle *et al.* (2012c) conduct a study on whether LDA topics make sense to practitioners (i.e., Mircosoft developers). The results show that although many LDA topics are perceptually valid, some topics are hard to understand. Hindle *et al.* (2012c) recommend that topics need to be interpreted, pruned, and labelled by experts for better understanding.

Hindle *et al.* (2011, 2012a) propose an approach to automatically generate more meaningful labels for LDA topics by classifying the topic labels using predefined labels derived from software engineering standards (ISO9126 standard of non-functional requirements). Labeled LDA (LLDA) (McIlroy *et al.*, 2015; Ramage *et al.*, 2009a) can also be used for generating more meaningful topics. However, LLDA requires a training set where the topics are annotated, which must be prepared manually.

In short, labelling and interpreting topics can be difficult and subjective, and may require much human effort. Future studies should explore ways to apply different approaches to automatically label the topics.

5.2.3 Data Preprocessing

Data preprocessing is a very important step before applying topic models. Studies (Madsen *et al.*, 2004; Thomas *et al.*, 2013) show that the data preprocessing steps have direct impact on the final topic models. Most common data preprocessing steps that we see are tokenization and stop word removal. We found that few studies report the use of pruning (15%). However, pruning can remove overly common (or rare) words, which may improve the performance of topic models. Note that since the words are related to the domain-level concepts in the system, filtering too many words may have negative effects on the performance. Future studies may want to explore how to best prune the words in order to achieve the optimal performance.

There are also no studies that use lemmatization. Lemmatization is similar to stemming, but lemmatization considers knowledge of the context, which may improve the accuracy. Future studies may want to apply lemmatization, or compare lemmatization with stemming when applying topic models on SE tasks.

5.2.4 Document Size

We found that many SE studies apply topic models at the method level (47%). However, topic models usually perform poorly on short documents (Jin *et al.*, 2011; Phan *et al.*, 2008a; Tang *et al.*, 2014), so the generated topics will be of low quality. We found that there are no MSR studies that discuss this problem nor ones that adapt one of the more advanced topic models for short documents. Future SE studies should consider using some of the specialized topic models for short documents (Guo and Diab, 2012; Jin *et al.*, 2011; Yan *et al.*, 2013) when applying topic models on smaller source code entities (e.g., methods).

5.2.5 Document Noise

When applying topic models for tasks such as analyzing software systems, common boilerplate code in each file may affect the final topics. The boilerplate code (or legal licensing text) may become one of the dominating topics in the system, but the topic is less interesting. Thus, depending on the use case, researchers may want to remove the boilerplate code or apply the topic evolution models discussed in Section 2.3.4.

5.2.6 Tendency to Employ More Complex Models

We found that many SE studies prefer using more complex topic models, such as LDA. Although LDA has the advantage of generating human-readable topics directly, simpler models such as LSI or even VSM may work better for some other tasks. Studies (Rao and Kak, 2011; Thomas *et al.*, 2013) have shown that simpler models outperform more complex models for tasks such as traceability linking and bug localization. In short, SE researchers may want to choose topic models that are more suitable for the task, instead of choosing more complex models.

Table 3 Summary of how surveyed articles apply topic models for different software engineering tasks. The numbers are shown in percentage for each category (i.e., *Topic Model*, *Repo. Used*, and *Evaluation*).

		clustering	feature	metrics	trend	traceability	debug	collections	other
Topic Model	LSI	69	55	43	23	89	45	14	0
	LDA	38	42	50	77	28	45	86	100
	Other	0	15	7	8	11	27	0	0
Repo. Used	source code	81	100	93	46	86	73	86	100
	email	0	0	0	0	3	0	0	0
	req./design	6	3	14	15	64	0	0	0
	logs	6	3	7	23	0	36	14	0
	bug reports	13	9	7	15	19	9	0	0
Evaluation	statistical	0	3	0	0	3	0	0	0
	task specific	50	52	57	31	92	100	14	56
	manual	13	9	7	15	11	18	14	0
	user study	0	0	0	8	3	0	0	0

5.2.7 Topic Consistency

Since topic models like LDA are based on probability distributions, different LDA runs may generate slightly different topics. This probabilistic behaviour may cause problems when combining different LDA runs (e.g., when studying software evolution). Therefore, researchers may want to consider using LDA models proposed by Mei and Zhai (2005), Hall *et al.* (2008), Linstead *et al.* (2008a), and Thomas *et al.* (2010) (Section 2.3.4) to maintain topic consistency across LDA runs.

5.2.8 Realistic Evaluation

Some probabilistic models like LDA are usually implemented using different sampling techniques, so the running time of the algorithm directly impacts the quality of the generated topics (Binkley *et al.*, 2014). However, researchers usually do not report the required efforts when applying topic models on software systems, which is an important measure for adapting the approach for practical uses. For example, taking over a week to run topic models on a system may not be feasible or practical if the system is constantly changing. Future studies should consider reporting the efforts (e.g., time) required to adapt the proposed approach.

6 Future Research Opportunities

In this Section, we discuss some opportunities for future work on applying topic models to software engineering tasks.

Underused Repositories From our survey study, we found that there remain several other software repositories that require more attention. For example, email archives and execution logs have rarely been studied, even though they are rich with information about a software system.

Underexplored Software Engineering Tasks Bug prediction, searching collections of software systems, and measuring the evolutionary trends of repositories are all underexplored tasks in the literature. In addition, traceability links are typically established between requirements documents and source code, although it would also be useful to find links between other repositories, such as emails and source code, and between source code documents themselves.

Additional Topic Models The variants of LDA listed in Section 2.3.3 have promising features that may directly improve the results of several software engineering tasks. For example, the correlated topic model, which models dependencies between topics, may allow sets of dependent topics in need of refactoring to be found in the source code. Additionally, the cross-collection topic model might allow similar topics to be discovered from the source code of related systems, such as Mozilla Firefox and Google Chrome. In addition, lightweight models such as BM25 or BM25F may be useful for bug localization or test case prioritization.

Data Preprocessing and Topic Model Parameters Prior study has shown the importance of data preprocessing and topic models parameters (Panichella *et al.*, 2013; Thomas *et al.*, 2013). Previous studies from the literature can be performed again after determining the best combination of the preprocessing steps and topic model parameters.

Additional Preprocessing Steps A preprocessing step that is currently less popular, but may also provide benefits, is query expansion (Carpineto and Romano, 2012), i.e., automatically fixing spelling errors, finding synonyms, or using or WordNet (Miller, 1995) to find related concepts and themes. Query expansion can be applied, for example, to bug localization datasets to reduce noise in the bug reports, and to help expand short or vague bug reports to provide more contexts. In addition, most preprocessing steps treat all words as equals, independent of their context. Considering context might allow the opportunity to give higher weights to important terms, technical terms, or system-specific terms. For example, it may be fruitful for the preprocessor to determine whether a bug report has an embedded code snippet and use this context to preserve identifier names in their entirety, so as to maximize the chance of linking the bug report to relevant source code entities that contain the same identifier names.

Special Topic Models for Software Data Topic models were initially proposed for natural language texts. Thus, topic models may not perform as well when applied on software development data (e.g., source code). We so far only see one study that proposes a new topic model that considers the properties of software data (Nguyen *et al.*, 2012). Future studies may want to propose new topic models that consider the structure of the software development data. Moreover, as we observed, many SE studies use task-specific evaluation approaches (e.g., measuring the number of discovered bugs). There could potentially be new general evaluation metrics for software engineering tasks. Such metrics could improve the future usage of topic modeling in SE, and possibly help improve general parameter tuning and specification.

Treating Software as Natural Language Recent work by Hindle *et al.* (2012b) has compared source code to natural language: both are created by humans, and while any given instance of either could theoretically be very complex, most of the time the instances are quite simple. The authors show that source code is indeed "natural", in that it is highly repetitive and predictable. As a consequence, models that deal with source code text can use this fact to construct more intelligent models.

7 Conclusion

The field of mining software repositories uses readily-available data to increase the productivity of developers and reduce project costs. Using all available data, both structured and unstructured, maximizes benefits. Since the majority of software repositories store unstructured data, researchers have used statistical topic models to mine textual information in the repositories. However, the performance of topic models is directly related to the usage and model parameters. Even though there are hundreds of studies on applying topic models to software repositories, there is no study that shows how the models are used in the software engineering research community, and which software engineering tasks are being supported through topic models. Knowing how researchers use the topic models may also help future studies improve the model performance.

In this paper, we surveyed 167 articles from the software engineering literature that use topic models. We found that:

- most studies focus on only a limited number of software engineering tasks;
- most studies use only basic topic models;
- and researchers usually treat topic models as black boxes without fully exploring their underlying assumptions and parameter values.

We have also provided possible direction of future work on applying topic models on software repositories. Our paper provides a starting point for new researchers who are interested in using topic models, and may help new researchers and practitioners determine how to best apply topic models to a particular software engineering task.

References

Abebe, S. L., Alicante, A., Corazza, A., and Tonella, P. (2013). Supporting concept location through identifier parsing and ontology extraction. *Journal of Systems Software*, **86**(11), 2919–2938.

Ahsan, S. N., Ferzund, J., and Wotawa, F. (2009). Automatic software bug triage system (BTS) based on Latent Semantic Indexing and Support Vector Machine. In *Proceedings of the 4th International Conference on Software Engineering Advances*, pages 216–221.

Alhindawi, N., Dragan, N., Collard, M., and Maletic, J. (2013a). Improving feature location by enhancing source code with stereotypes. In *Proceedings of the 2013 29th IEEE International Conference on Software Maintenance*, pages 300–309.

Alhindawi, N., Meqdadi, O., Bartman, B., and Maletic, J. (2013b). A tracelab-based solution for identifying traceability links using lsi. In *Proceedings of 2013 International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, pages 79–82.

Ali, N., Sabane, A., Gueheneuc, Y., and Antoniol, G. (2012). Improving bug location using binary class relationships. In *Proceedings of the 2012 IEEE 12th International Working Conference on Source Code Analysis and Manipulation*, pages 174–183.

Ali, N., Sharafi, Z., Gueheneuc, Y.-G., and Antoniol, G. (2014). An empirical study on the importance of source code entities for requirements traceability. *Empirical Softw. Engg.*

Alipour, A., Hindle, A., and Stroulia, E. (2013). A contextual approach towards more accurate duplicate bug report detection. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 183–192.

Allamanis, M. and Sutton, C. (2013). Why, when, and what: Analyzing Stack Overflow questions by topic, type, and code. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 53–56.

Andrzejewski, D., Mulhern, A., Liblit, B., and Zhu, X. (2007). Statistical debugging using latent topic models. In *Proceedings of the 18th European Conference on Machine Learning*, pages 6–17.

Anthes, G. (2010). Topic models vs. unstructured data. *Communications of the ACM*, **53**, 16–18.

Antoniol, G., Hayes, J. H., Gueheneuc, Y. G., and Di Penta, M. (2008). Reuse or rewrite: Combining textual, static, and dynamic analyses to assess the cost of keeping a system up-to-date. In *Proceedings of the 24th International Conference on Software Maintenance*, pages 147–156.

Asadi, F., Antoniol, G., and Gueheneuc, Y.-G. (2010a). Concept location with genetic algorithms: A comparison of four distributed architectures. In *Proceedings of the 2Nd International Symposium on Search Based Software Engineering*, SSBSE '10, pages 153–162.

Asadi, F., Di Penta, M., Antoniol, G., and Gueheneuc, Y.-G. (2010b). A heuristic-based approach to identify concepts in execution traces. In *Proceedings of the 2010 14th European Conference on Software Maintenance and Reengineering (CSMR)*, pages 31–40.

Asuncion, H. U., Asuncion, A. U., and Taylor, R. N. (2010). Software traceability with topic modeling. In *Proceedings of the 32nd International Conference on Software Engineering*, pages 95–104.

Baeza-Yates, R. and Ribeiro-Neto, B. (1999). *Modern information retrieval*, volume 463. ACM press New York.

Bajaj, K., Pattabiraman, K., and Mesbah, A. (2014). Mining questions asked by web developers. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 112–121.

Bajracharya, S. and Lopes, C. (2009). Mining search topics from a code search engine usage log. In *Proceedings of the 6th International Working Conference on Mining Software Repositories*, pages 111–120.

Bajracharya, S. K. and Lopes, C. V. (2010). Analyzing and mining a code search engine usage log. *Empirical Software Engineering*, pages 1–43.

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

Barnard, K., Duygulu, P., Forsyth, D., De Freitas, N., Blei, D. M., and Jordan, M. I. (2003). Matching words and pictures. *The Journal of Machine Learning Research*, **3**, 1107–1135.

Bartholomew, D. J. (1987). *Latent variable models and factors analysis*. Oxford University Press, Inc., New York, NY, USA.

Barua, A., Thomas, S. W., and Hassan, A. E. (2012). What are developers talking about? An analysis of topics and trends in Stack Overflow. *Empirical Software Engineering*.

Bassett, B. and Kraft, N. (2013). Structural information based term weighting in text retrieval for feature location. In *Proceedings of the 2013 21st IEEE International Conference on Program Comprehension*, pages 133–141.

Bavota, G., De Lucia, A., Marcus, A., and Oliveto, R. (2010). A two-step technique for extract class refactoring. In *Proceedings of the 25th International Conference on Automated Software Engineering*, pages 151–154.

Bavota, G., Lucia, A. D., Marcus, A., and Oliveto, R. (2012). Using structural and semantic measures to improve software modularization. *Empirical Softw. Engg.*

Bavota, G., De Lucia, A., Oliveto, R., Panichella, A., Ricci, F., and Tortora, G. (2013). The role of artefact corpus in lsi-based traceability recovery. In *Proceedings of the 2013 International Workshop on Traceability in Emerging Forms of Software Engineering*, pages 83–89.

Bavota, G., Oliveto, R., Gethers, M., Poshyvanyk, D., and De Lucia, A. (2014). Methodbook: Recommending move method refactorings via relational topic models. *IEEE Transactions on Software Engineering*, **40**(7), 671–694.

Beard, M., Kraft, N., Etzkorn, L., and Lukins, S. (2011). Measuring the accuracy of information retrieval based bug localization techniques. In *Proceedings of the 2011 18th Working Conference on Reverse Engineering*, WCRE '11, pages 124–128.

Bellon, S., Koschke, R., Antoniol, G., Krinke, J., and Merlo, E. (2007). Comparison and evaluation of clone detection tools. *IEEE Transactions on Software Engineering*, pages 577–591.

Bettenburg, N. and Adams, B. (2010). Workshop on Mining Unstructured Data (MUD) because "Mining Unstructured Data is like fishing in muddy waters"! In *Proceedings of the 17th Working Conference on Reverse Engineering*, pages 277–278.

Biggers, L. R., Bocovich, C., Capshaw, R., Eddy, B. P., Etzkorn, L. H., and Kraft, N. A. (2014). Configuring latent dirichlet allocation based feature location. *Empirical Softw. Engg.*, **19**(3), 465–500.

Binkley, D., Lawrie, D., and Uehlinger, C. (2012). Vocabulary normalization improves ir-based concept location. In *Proceedings of the 2012 28th IEEE International Conference on Software Maintenance*, pages 588–591.

Binkley, D., Heinz, D., Lawrie, D., and Overfelt, J. (2014). Understanding lda in source code analysis. In *Proceedings of the 22nd International Conference on Program Comprehension*, ICPC 2014, pages 26–36.

Bishop, C. M. (1998). Latent variable models. *Learning in graphical models*.

Blei, D., Griffiths, T. L., Jordan, M. I., and Tenenbaum, J. B. (2004). Hierarchical topic models and the nested Chinese restaurant process. *Advances in Neural Information Processing Systems*, **16**, 106.

Blei, D. M. and Lafferty, J. D. (2006). Dynamic topic models. In *Proceedings of the 23rd international conference on Machine learning*, pages 113–120. ACM.

Blei, D. M. and Lafferty, J. D. (2007). A correlated topic model of science. *Annals of Applied Statistics*, **1**(1), 17–35.

Blei, D. M. and Lafferty, J. D. (2009). Topic models. In *Text Mining: Classification, Clustering, and Applications*, pages 71–94. Chapman & Hall, London, UK.

Blei, D. M. and McAuliffe, J. (2008). Supervised topic models. *Advances in Neural Information Processing Systems*, **20**, 121–128.

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent Dirichlet allocation. *Journal of Machine Learning Research*, **3**, 993–1022.

Blei, D. M., Griffiths, T. L., and Jordan, M. I. (2010). The nested chinese restaurant process and bayesian nonparametric inference of topic hierarchies. *Journal of the ACM*, **57**(2), 1–30.

Blumberg, R. and Atre, S. (2003). The problem with unstructured data. *DM Review*, **13**, 42–49.

Borg, M., Runeson, P., and Ard, A. (2014). Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability. *Empirical Software Engineering*, **19**(6), 1565–1616.

Bose, J. C. and Suresh, U. (2008). Root cause analysis using sequence alignment and Latent Semantic Indexing. In *Proceedings of the 19th Australian Conference on Software Engineering*, pages 367–376.

Bradford, R. B. (2008). An empirical study of required dimensionality for large-scale latent semantic indexing applications. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, CIKM '08, pages 153–162.

Brickey, J., Walczak, S., and Burgess, T. (2012). Comparing semi-automated clustering methods for persona development. *IEEE Trans. Softw. Eng.*, **38**(3), 537–546.

Campbell, J.-C., Zhang, C., Xu, Z., Hindle, A., and Miller, J. (2013). Deficient documentation detection a methodology to locate deficient project documentation using topic analysis. In *Proceedings of the 2013 10th IEEE Working Conference on Mining Software Repositories*, pages 57–60.

Canfora, G., Cerulo, L., Cimitile, M., and Di Penta, M. (2014). How changes affect software entropy: An empirical study. *Empirical Softw. Engg.*, **19**(1), 1–38.

Capobianco, G., De Lucia, A., Oliveto, R., Panichella, A., and Panichella, S. (2009). Traceability recovery using numerical analysis. In *Proceedings of the 16th Working Conference on Reverse Engineering*, pages 195–204.

Carpineto, C. and Romano, G. (2012). A survey of automatic query expansion in information retrieval. *ACM Computing Surveys*, **44**(1), 1–50.

Chang, J. and Blei, D. M. (2009). Relational topic models for document networks. *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics*, pages 81–88.

Chang, J., Boyd-Graber, J., and Blei, D. M. (2009). Connections between the lines: Augmenting social networks with text. In *Proceedings of the 15th International Conference on Knowledge Discovery and Data Mining*, pages 169–178.

Chen, T.-H., Thomas, S. W., Nagappan, M., and Hassan, A. E. (2012). Explaining software defects using topic models. In *Proceedings of the 2012 9th IEEE Working Conference on Mining Software Repositories*, MSR '12, pages 189–198.

Chen, T.-H., Thomas, S. W., Hemmati, H., Nagappan, M., and Hassan, A. E. (2015). An empirical study on topic defect-proneness and testedness. In *under submission*.

Cleary, B., Exton, C., Buckley, J., and English, M. (2008). An empirical analysis of information retrieval based concept location techniques in software comprehension. *Empirical Software Engineering*, **14**(1), 93–130.

Comon, P. (1994). Independent component analysis, a new concept? *Signal processing*, **36**(3), 287–314.

Corley, C., Kammer, E., and Kraft, N. (2012). Modeling the ownership of source code topics. In *Proceedings of the 2012 IEEE 20th International Conference on Program Comprehension*, pages 173–182.

Dallmeier, V. and Zimmermann, T. (2007). Extraction of bug localization benchmarks from history. In *Proceedings of the 22nd International Conference on Automated Software Engineering*, pages 433–436.

Dasgupta, T., Grechanik, M., Moritz, E., Dit, B., and Poshyvanyk, D. (2013). Enhancing software traceability by automatically expanding corpora with relevant documentation. In *Proceedings of the 2013 IEEE International Conference on Software Maintenance*, ICSM '13, pages 320–329.

de Boer, R. C. and van Vliet, H. (2008). Architectural knowledge discovery with Latent Semantic Analysis: constructing a reading guide for software product audits. *Journal of Systems and Software*, **81**(9), 1456–1469.

De Lucia, A., Fasano, F., Oliveto, R., and Tortora, G. (2004). Enhancing an artefact management system with traceability recovery features. In *Proceedings of the 20th International Conference on Software Maintenance*, pages 306–315.

De Lucia, A., Fasano, F., Oliveto, R., and Tortora, G. (2006). Can information retrieval techniques effectively support traceability link recovery? In *Proceedings of the 14th International Conference on Program Comprehension*, pages 307–316.

De Lucia, A., Fasano, F., Oliveto, R., and Tortora, G. (2007). Recovering traceability links in software artifact management systems using information retrieval methods. *ACM Transactions on Software Engineering and Methodology*, **16**(4).

De Lucia, A., Di Penta, M., Oliveto, R., Panichella, A., and Panichella, S. (2011). Improving ir-based traceability recovery using smoothing filters. In *Proceedings of the 2011 IEEE 19th International Conference on Program Comprehension*, pages 21–30.

De Lucia, A., Di Penta, M., Oliveto, R., Panichella, A., and Panichella, S. (2012). Using ir methods for labeling source code artifacts: Is it worthwhile? In *Proceedings of the 2012 20th International Conference on Program Comprehension*, pages 193–202.

De Lucia, A., Di Penta, M., Oliveto, R., Panichella, A., and Panichella, S. (2014). Labeling source code with information retrieval methods: an empirical study. *Empirical Software Engineering*, pages 1–38.

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science*, **41**(6), 391–407.

Dietterich, T. G. (2000). Ensemble methods in machine learning. In *Proceedings of the First International Workshop on Multiple Classifier Systems*, MCS '00, pages 1–15.

Dit, B., Poshyvanyk, D., and Marcus, A. (2008). Measuring the semantic similarity of comments in bug reports. In *Proceedings 1st International Workshop on Semantic Technologies in System Maintenance*.

Dit, B., Panichella, A., Moritz, E., Oliveto, R., Di Penta, M., Poshyvanyk, D., and De Lucia, A. (2013a). Configuring topic models for software engineering tasks in tracelab. In *Proceedings of the 2013 International Workshop on Traceability in Emerging Forms of Software Engineering*, pages 105–109.

Dit, B., Revelle, M., and Poshyvanyk, D. (2013b). Integrating information retrieval, execution and link analysis algorithms to improve feature location in software. *Empirical Softw. Engg.*, **18**(2), 277–309.

Dit, B., Moritz, E., Linares-Vsquez, M., and Poshyvanyk, D. (2013c). Supporting and accelerating reproducible research in software maintenance using tracelab component library. In *Proceedings of the 2013 IEEE International Conference on Software Maintenance*, ICSM '13, pages 330–339.

Dit, B., Moritz, E., Linares-Vsquez, M., Poshyvanyk, D., and Cleland-Huang, J. (2014). Supporting and accelerating reproducible empirical research in software evolution and maintenance using tracelab component library. *Empirical Software Engineering*, pages 1–39.

Eddy, B., Robinson, J., Kraft, N., and Carver, J. (2013). Evaluating source code summarization techniques: Replication and expansion. In *Proceedings of 2013 IEEE 21st International Conference on Program Comprehension*, pages 13–22.

Eyal-Salman, H., Seriai, A.-D., and Dony, C. (2013). Feature-to-code traceability in legacy software variants. In *Proceedings of 2013 39th EUROMICRO Conference on the Software Engineering and Advanced Applications*, pages 57–61.

Flaherty, P., Giaever, G., Kumm, J., Jordan, M. I., and Arkin, A. P. (2005). A latent variable model for chemogenomic profiling. *Bioinformatics*, **21**(15), 3286.

Gall, C. S., Lukins, S., Etzkorn, L., Gholston, S., Farrington, P., Utley, D., Fortune, J., and Virani, S. (2008). Semantic software metrics computed from natural language design specifications. *Software, IET*, **2**(1), 17–26.

Galvis Carreño, L. V. and Winbladh, K. (2013). Analysis of user comments: An approach for software requirements evolution. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 582–591.

Gamma, E. (2007). JHotDraw. http://www.jhotdraw.org/.

Gethers, M. and Poshyvanyk, D. (2010). Using relational topic models to capture coupling among classes in object-oriented software systems. In *Proceedings of the 26th International Conference on Software Maintenance*, pages 1–10.

Gethers, M., Kagdi, H., Dit, B., and Poshyvanyk, D. (2011a). An adaptive approach to impact analysis from change requests to source code. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, pages 540–543.

Gethers, M., Savage, T., Di Penta, M., Oliveto, R., Poshyvanyk, D., and De Lucia, A. (2011b). Codetopics: Which topic am i coding now? In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 1034–1036.

Gethers, M., Oliveto, R., Poshyvanyk, D., and Lucia, A. (2011c). On integrating orthogonal information retrieval methods to improve traceability recovery. In *Proceedings of the 27th International Conference on Software Maintenance*, pages 133–142.

Gethers, M., Oliveto, R., Poshyvanyk, D., and Lucia, A. D. (2011d). On integrating orthogonal information retrieval methods to improve traceability recovery. In

Proceedings of the 2011 27th International Conference on Software Maintenance, ICSM '11, pages 133–142.

Gethers, M., Dit, B., Kagdi, H., and Poshyvanyk, D. (2012). Integrated impact analysis for managing software changes. In *Proceedings of the 2012 34th International Conference on Software Engineering*, pages 430–440.

Godfrey, M. W., Hassan, A. E., Herbsleb, J., Murphy, G. C., Robillard, M., Devanbu, P., Mockus, A., Perry, D. E., and Notkin, D. (2008). Future of mining software archives: A roundtable. *IEEE Software*, **26**(1), 67–70.

Gorla, A., Tavecchia, I., Gross, F., and Zeller, A. (2014). Checking app behavior against app descriptions. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 1025–1035.

Grant, S. and Cordy, J. (2014). Examining the relationship between topic model similarity and software maintenance. In *Proceedings of the 2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE)*, pages 303–307.

Grant, S. and Cordy, J. R. (2009). Vector space analysis of software clones. In *Proceedings of the 17th International Conference on Program Comprehension*, pages 233–237.

Grant, S. and Cordy, J. R. (2010). Estimating the optimal number of latent concepts in source code analysis. In *Proceedings of the 10th International Working Conference on Source Code Analysis and Manipulation*, pages 65–74.

Grant, S., Cordy, J. R., and Skillicorn, D. (2008). Automated concept location using independent component analysis. In *Proceedings of the 15th Working Conference on Reverse Engineering*, pages 138–142.

Grant, S., Martin, D., Cordy, J., and Skillicorn, D. (2011a). Contextualized semantic analysis of web services. In *Proceedings of the 2011 13th IEEE International Symposium on Web Systems Evolution*, pages 33–42.

Grant, S., Cordy, J. R., and Skillicorn, D. B. (2011b). Reverse engineering comaintenance relationships using conceptual analysis of source code. In *Proceedings of the 2011 18th Working Conference on Reverse Engineering*, WCRE '11, pages 87–91.

Grant, S., Cordy, J., and Skillicorn, D. (2012). Using topic models to support software maintenance. In *Proceedings of 2012 16th European Conference on the Software Maintenance and Reengineering (CSMR)*, pages 403–408.

Grechanik, M., Fu, C., Xie, Q., McMillan, C., Poshyvanyk, D., and Cumby, C. (2010). A search engine for finding highly relevant applications. In *Proceedings of the 32nd International Conference on Software Engineering*, pages 475–484.

Griffiths, T. L. and Steyvers, M. (2004). Finding scientific topics. *Proceedings of the National Academy of Sciences*, **101**, 5228–5235.

Griffiths, T. L., Steyvers, M., and Tenenbaum, J. B. (2007). Topics in semantic representation. *Psychological Review*, **114**(2), 211–244.

Grimes, S. (2008). Unstructured data and the 80 percent rule. *Clarabridge Bridgepoints*.

Guo, W. and Diab, M. (2012). Modeling sentences in the latent space. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers - Volume 1*, ACL '12, pages 864–872.

Haiduc, S., Bavota, G., Marcus, A., Oliveto, R., De Lucia, A., and Menzies, T. (2013). Automatic query reformulations for text retrieval in software engineering. In *Proceedings of the 2013 International Conference on Software Engineer-*

ing, ICSE '13, pages 842–851.

Hall, D., Jurafsky, D., and Manning, C. D. (2008). Studying the history of ideas using topic models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 363–371. ACL.

Han, D., Zhang, C., Fan, X., Hindle, A., Wong, K., and Stroulia, E. (2012). Understanding android fragmentation with topic analysis of vendor-specific bugs. In *Proceedings of the 2012 19th Working Conference on Reverse Engineering*, WCRE '12, pages 83–92.

Hassan, A. E. (2004). *Mining Software Repositories to Assist Developers and Support Managers*. Ph.D. thesis, University of Waterloo, Waterloo, ON, Canada.

Hassan, A. E. (2008). The road ahead for mining software repositories. In *Frontiers of Software Maintenance*, pages 48–57.

Hassan, A. E. and Holt, R. C. (2005). The top ten list: Dynamic fault prediction. In *Proceedings of the 21st International Conference on Software Maintenance*, pages 263–272.

Hassan, A. E. and Xie, T. (2010). Software intelligence: The future of mining software engineering data. In *Proceedings of the FSE/SDP workshop on Future of Software Engineering Research*, pages 161–166.

Hayes, J. H., Dekhtyar, A., and Sundaram, S. K. (2006). Advancing candidate link generation for requirements tracing: The study of methods. *IEEE Transactions on Software Engineering*, pages 4–19.

Hindle, A., Godfrey, M. W., and Holt, R. C. (2009). What's hot and what's not: Windowed developer topic analysis. In *Proceedings of the 25th International Conference on Software Maintenance*, pages 339–348.

Hindle, A., Godfrey, M. W., and Holt, R. C. (2010). Software process recovery using recovered unified process views. In *Proceedings of the 26th International Conference on Software Maintenance*, pages 1–10.

Hindle, A., Ernst, N. A., Godfrey, M. W., and Mylopoulos, J. (2011). Automated topic naming to support cross-project analysis of software maintenance activities. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, MSR '11, pages 163–172.

Hindle, A., Ernst, N. A., Godfrey, M. W., and Mylopoulos, J. (2012a). Automated topic naming - supportin cross-project analysis of software maintenance activities. *Empirical Softw. Engg.*

Hindle, A., Barr, E. T., Su, Z., Gabel, M., and Devanbu, P. T. (2012b). On the naturalness of software. In *Proceedings of the 34th International Conference on Software Engineering*, pages 837–847.

Hindle, A., Bird, C., Zimmermann, T., and Nagappan, N. (2012c). Relating requirements to implementation via topic analysis: Do topics extracted from requirements make sense to managers and developers? In *Proceedings of the 2012 28th IEEE International Conference on Software Maintenance*, pages 243–252.

Hindle, A., Bird, C., Zimmermann, T., and Nagappan, N. (2014). Do topics make sense to managers and developers? *Empirical Softw. Engg.*

Hofmann, T. (1999). Probabilistic Latent Semantic Indexing. In *Proceedings of the 22nd International Conference on Research and Development in Information Retrieval*, pages 50–57.

Hofmann, T. (2001). Unsupervised learning by probabilistic Latent Semantic Analysis. *Machine Learning*, **42**(1), 177–196.

Hu, W. and Wong, K. (2013). Using citation influence to predict software defects. In *Proceedings of the 2013 10th IEEE Working Conference on Mining Software Repositories*, pages 419–428.

Hu, X., Cai, Z., Franceschetti, D., Penumatsa, P., and Graesser, A. C. (2003). LSA: The first dimension and dimensional weighting. In *Proceedings of the 25th Meeting of the Cognitive Science Society*, pages 1–6, Boston. Cognitive Science Society.

Iacob, C. and Harrison, R. (2013). Retrieving and analyzing mobile apps feature requests from online reviews. In *Proceedings of the 2013 10th IEEE Working Conference on Mining Software Repositories*, pages 41–44.

Islam, M., Marchetto, A., Susi, A., Kessler, F., and Scanniello, G. (2012a). Motcp: A tool for the prioritization of test cases based on a sorting genetic algorithm and latent semantic indexing. In *Proceedings of the 2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pages 654–657.

Islam, M., Marchetto, A., Susi, A., and Scanniello, G. (2012b). A multi-objective technique to prioritize test cases based on latent semantic indexing. In *Proceedings of the 2012 16th European Conference on Software Maintenance and Reengineering*, pages 21–30.

Jiang, H., Nguyen, T. N., Chen, I., Jaygarl, H., and Chang, C. (2008). Incremental Latent Semantic Indexing for automatic traceability link evolution management. In *Proceedings of the 23rd International Conference on Automated Software Engineering*, pages 59–68.

Jin, O., Liu, N. N., Zhao, K., Yu, Y., and Yang, Q. (2011). Transferring topical knowledge from auxiliary long texts for short text clustering. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, pages 775–784.

Jolliffe, I. (2002). *Principal component analysis*. Springer, New York.

Kagdi, H., Gethers, M., Poshyvanyk, D., and Collard, M. (2010). Blending conceptual and evolutionary couplings to support change impact analysis in source code. In *Proceedings of the 17th Working Conference on Reverse Engineering*, pages 119–128.

Kagdi, H., Gethers, M., Poshyvanyk, D., and Hammad, M. (2012a). Assigning change requests to software developers. *Journal of Software: Evolution and Process*, **24**(1), 3–33.

Kagdi, H., Gethers, M., and Poshyvanyk, D. (2012b). Integrating conceptual and logical couplings for change impact analysis in software. *Empirical Softw. Engg.*

Kamei, Y., Matsumoto, S., Monden, A., Matsumoto, K., Adams, B., and Hassan, A. E. (2010). Revisiting common bug prediction findings using effort-aware models. In *Proceedings of the 26th International Conference on Software Maintenance*, pages 1–10.

Kaushik, N. and Tahvildari, L. (2012). A comparative study of the performance of ir models on duplicate bug detection. In *Proceedings of the 2012 16th European Conference on Software Maintenance and Reengineering*, CSMR '12, pages 159–168.

Kaushik, N., Tahvildari, L., and Moore, M. (2011). Reconstructing traceability between bugs and test cases: An experimental study. In *Proceedings of the 2011 18th Working Conference on Reverse Engineering*, WCRE '11, pages 411–414.

Kawaguchi, S., Garg, P. K., Matsushita, M., and Inoue, K. (2006). Mudablue: An automatic categorization system for open source repositories. *Journal of*

Systems and Software, **79**(7), 939–953.

Kelly, M., Alexander, J., Adams, B., and Hassan, A. (2011). Recovering a balanced overview of topics in a software domain. In *Proceedings of the 2011 11th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 135–144.

Kitchenham, B. A., Budgen, D., and Pearl Brereton, O. (2011). Using mapping studies as the basis for further research - a participant-observer case study. *Inf. Softw. Technol.*, **53**(6), 638–651.

Kouters, E., Vasilescu, B., Serebrenik, A., and van den Brand, M. (2012). Who's who in gnome: Using lsa to merge software repository identities. In *Proceedings of the 2012 28th IEEE International Conference on Software Maintenance*, pages 592–595.

Kuhn, A., Ducasse, S., and Girba, T. (2005). Enriching reverse engineering with semantic clustering. In *Proceedings of the 12th Working Conference on Reverse Engineering*, pages 133–142.

Kuhn, A., Ducasse, S., and Girba, T. (2007). Semantic clustering: Identifying topics in source code. *Information and Software Technology*, **49**(3), 230–243.

Kuhn, A., Loretan, P., and Nierstrasz, O. (2008). Consistent layout for thematic software maps. In *Proceedings of the 15th Working Conference on Reverse Engineering*, pages 209–218.

Kuhn, A., Erni, D., Loretan, P., and Nierstrasz, O. (2010). Software cartography: Thematic software visualization with consistent layout. *Journal of Software Maintenance and Evolution: Research and Practice*, pages 191–210.

Le, T.-D., Wang, S., and Lo, D. (2013). Multi-abstraction concern localization. In *Proceedings of the 2013 29th IEEE International Conference on Software Maintenance (ICSM)*, pages 364–367.

Lehman, M. M. (1980). Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, **68**(9), 1060–1076.

Li, W. and McCallum, A. (2006). Pachinko allocation: DAG-structured mixture models of topic correlations. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 577–584.

Limsettho, N., Hata, H., and Matsumoto, K.-I. (2014). Comparing hierarchical dirichlet process with latent dirichlet allocation in bug report multiclass classification. In *Proceedings of 2014 15th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pages 1–6.

Lin, M. Y., Amor, R., and Tempero, E. (2006). A Java reuse repository for Eclipse using LSI. In *Proceedings of the 2006 Australian Software Engineering Conference*, pages 351–362.

Linares-Vásquez, M., Dit, B., and Poshyvanyk, D. (2013). An exploratory analysis of mobile development issues using Stack Overflow. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 93–96.

Linstead, E. and Baldi, P. (2009). Mining the coherence of GNOME bug reports with statistical topic models. In *Proceedings of the 6th Working Conference on Mining Software Repositories*, pages 99–102.

Linstead, E., Rigor, P., Bajracharya, S., Lopes, C., and Baldi, P. (2007a). Mining concepts from code with probabilistic topic models. In *Proceedings of the 22nd International Conference on Automated Software Engineering*, pages 461–464.

Linstead, E., Rigor, P., Bajracharya, S., Lopes, C., and Baldi, P. (2007b). Mining Eclipse developer contributions via author-topic models. In *Proceedings of the 4th International Workshop on Mining Software Repositories*, pages 30–33.

Linstead, E., Lopes, C., and Baldi, P. (2008a). An application of latent Dirichlet allocation to analyzing software evolution. In *Proceedings of the 7th International Conference on Machine Learning and Applications*, pages 813–818.

Linstead, E., Rigor, P., Bajracharya, S., Lopes, C., and Baldi, P. (2008b). Mining internet-scale software repositories. In *Advances in Neural Information Processing Systems*, volume 2007, pages 929–936.

Linstead, E., Bajracharya, S., Ngo, T., Rigor, P., Lopes, C., and Baldi, P. (2008c). Sourcerer: mining and searching internet-scale software repositories. *Data Mining and Knowledge Discovery*, **18**(2), 300–336.

Linstead, E., Hughes, L., Lopes, C., and Baldi, P. (2009). Software analysis with unsupervised topic models. In *NIPS Workshop on Application of Topic Models: Text and Beyond*.

Liu, Y., Poshyvanyk, D., Ferenc, R., Gyimothy, T., and Chrisochoides, N. (2009). Modeling class cohesion as mixtures of latent topics. In *Proceedings of the 25th International Conference on Software Maintenance*, pages 233–242.

Loehlin, J. C. (1987). *Latent variable models*. Erlbaum Hillsdale, NJ.

Lohar, S., Amornborvornwong, S., Zisman, A., and Cleland-Huang, J. (2013). Improving trace accuracy through data-driven configuration and composition of tracing features. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, pages 378–388.

Lormans, M. (2007). Monitoring requirements evolution using views. In *Proceedings of the 11th European Conference on Software Maintenance and Reengineering*, pages 349–352.

Lormans, M. and Van Deursen, A. (2006). Can LSI help reconstructing requirements traceability in design and test? In *Proceedings of 10th European Conference on Software Maintenance and Reengineering*, pages 47–56.

Lormans, M., Gross, H. G., van Deursen, A., and van Solingen, R. (2006). Monitoring requirements coverage using reconstructed views: An industrial case study. In *Proceedings of the 13th Working Conference on Reverse Engineering*, pages 275–284.

Lormans, M., Deursen, A., and Gross, H.-G. (2008). An industrial case study in reconstructing requirements views. *Empirical Softw. Engg.*, **13**(6), 727–760.

Lukins, S. K., Kraft, N. A., and Etzkorn, L. H. (2008). Source code retrieval for bug localization using latent Dirichlet allocation. In *Proceedings of the 15th Working Conference on Reverse Engineering*, pages 155–164.

Lukins, S. K., Kraft, N. A., and Etzkorn, L. H. (2010). Bug localization using latent Dirichlet allocation. *Information and Software Technology*, **52**(9), 972–990.

Madsen, R., Sigurdsson, S., Hansen, L., and Larsen, J. (2004). Pruning the vocabulary for better context recognition. In *Proceedings of the 17th International Conference on Pattern Recognition*, pages 483–488.

Maletic, J. I. and Marcus, A. (2001). Supporting program comprehension using semantic and structural information. In *Proceedings of the 23rd International Conference on Software Engineering*, pages 103–112.

Maletic, J. I. and Valluri, N. (1999). Automatic software clustering via Latent Semantic Analysis. In *Proceeding of the 14th International Conference on Automated Software Engineering*, pages 251–254.

Manning, C. D., Raghavan, P., and Schutze, H. (2008). *Introduction to information retrieval*, volume 1. Cambridge University Press Cambridge, UK.

Marcus, A. (2004). Semantic driven program analysis. In *Proceedings of the 20th International Conference on Software Maintenance*, pages 469–473.

Marcus, A. and Maletic, J. I. (2001). Identification of high-level concept clones in source code. In *Proceedings of the 16th International Conference on Automated Software Engineering*, pages 107–114.

Marcus, A. and Maletic, J. I. (2003). Recovering documentation-to-source-code traceability links using Latent Semantic Indexing. In *Proceedings of the 25th International Conference on Software Engineering*, pages 125–135.

Marcus, A., Sergeyev, A., Rajlich, V., and Maletic, J. I. (2004). An information retrieval approach to concept location in source code. In *Proceedings of the 11th Working Conference on Reverse Engineering*, pages 214–223.

Marcus, A., Rajlich, V., Buchta, J., Petrenko, M., and Sergeyev, A. (2005). Static techniques for concept location in object-oriented code. In *Proceedings of the 13th International Workshop on Program Comprehension*, pages 33–42.

Marcus, A., Poshyvanyk, D., and Ferenc, R. (2008). Using the conceptual cohesion of classes for fault prediction in object-oriented systems. *IEEE Transactions on Software Engineering*, **34**(2), 287–300.

Maskeri, G., Sarkar, S., and Heafield, K. (2008). Mining business topics in source code using latent Dirichlet allocation. In *Proceedings of the 1st conference on India software engineering conference*, pages 113–120.

McCallum, A. K. (2002). Mallet: A machine learning for language toolkit. http://mallet.cs.umass.edu.

McIlroy, S., Ali, N., Khalid, H., and E. Hassan, A. (2015). Analyzing and automatically labelling the types of user issues that are raised in mobile app reviews. *Empirical Software Engineering*, pages 1–40.

McMillan, C., Poshyvanyk, D., and Revelle, M. (2009). Combining textual and structural analysis of software artifacts for traceability link recovery. In *Proceedings of the ICSE Workshop on Traceability in Emerging Forms of Software Engineering*, pages 41–48.

Medini, S. (2011). Scalable automatic concept mining from execution traces. In *Proceedings of the 2011 19th International Conference on Program Comprehension (ICPC)*, pages 238–241.

Medini, S., Antoniol, G., Gueheneuc, Y., Di Penta, M., and Tonella, P. (2012). Scan: An approach to label and relate execution trace segments. In *Proceedings of 2012 19th Working Conference on the Reverse Engineering (WCRE)*, pages 135–144.

Mei, Q. and Zhai, C. X. (2005). Discovering evolutionary theme patterns from text: an exploration of temporal text mining. In *Proceedings of the 11th International Conference on Knowledge Discovery in Data Mining*, pages 198–207.

Mei, Q., Shen, X., and Zhai, C. X. (2007). Automatic labeling of multinomial topic models. In *Proceedings of the 13th international conference on Knowledge discovery and data mining*, pages 490–499.

Mei, Q., Cai, D., Zhang, D., and Zhai, C. X. (2008). Topic modeling with network regularization. In *Proceeding of the 17th International Conference on World Wide Web*, pages 101–110.

Miller, G. A. (1995). WordNet: A lexical database for english. *Communications of the ACM*, **38**(11), 39–41.

Mimno, D., Wallach, H. M., Naradowsky, J., Smith, D. A., and McCallum, A. (2009). Polylingual topic models. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 880–889.

Misirli, A. T., Bener, A. B., and Turhan, B. (2011). An industrial case study of classifier ensembles for locating software defects. *Software Quality Journal*, **19**(3), 515–536.

Misra, J. and Das, S. (2013). Entity disambiguation in natural language text requirements. In *Proceedings of the 2013 20th Asia-Pacific Software Engineering Conference (APSEC)*, volume 1, pages 239–246.

Misra, J., Annervaz, K. M., Kaulgud, V., Sengupta, S., and Titus, G. (2012). Software clustering: Unifying syntactic and semantic features. In *Proceedings of the 2012 19th Working Conference on Reverse Engineering*, pages 113–122.

Moritz, E., Linares-Vasquez, M., Poshyvanyk, D., Grechanik, M., McMillan, C., and Gethers, M. (2013). Export: Detecting and visualizing api usages in large source code repositories. In *Proceedings of the 2013 IEEE/ACM 28th International Conference on Automated Software Engineering*, pages 646–651.

Naguib, H., Narayan, N., Brugge, B., and Helal, D. (2013). Bug report assignee recommendation using activity profiles. In *Proceedings of the 2013 10th IEEE Working Conference on Mining Software Repositories*, pages 22–30.

Neuhaus, S. and Zimmermann, T. (2010). Security trend analysis with CVE topic models. In *Proceedings of the 21st International Symposium on Software Reliability Engineering*, pages 111–120.

Newman, D., Lau, J. H., Grieser, K., and Baldwin, T. (2010). Automatic evaluation of topic coherence. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 100–108.

Nguyen, A., Nguyen, T. T., Nguyen, T., Lo, D., and Sun, C. (2012). Duplicate bug report detection with a combination of information retrieval and topic modeling. In *Proceedings of the 2012 27th IEEE/ACM International Conference on Automated Software Engineering*, pages 70–79.

Nguyen, A. T., Nguyen, T. T., Al-Kofahi, J., Nguyen, H. V., and Nguyen, T. N. (2011a). A topic-based approach for narrowing the search space of buggy files from a bug report. In *Proceedings of the 26th International Conference on Automated Software Engineering*, pages 263–272.

Nguyen, T. H., Adams, B., and Hassan, A. E. (2010). A case study of bias in bug-fix datasets. In *Proceedings of the 17th Working Conference on Reverse Engineering*, pages 259–268.

Nguyen, T. T., Nguyen, T. N., and Phuong, T. M. (2011b). Topic-based defect prediction (nier track). In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 932–935.

Nie, K. and Zhang, L. (2012). Software feature location based on topic models. In *Proceedings of the 2012 19th Asia-Pacific Software Engineering Conference*, APSEC '12, pages 547–552.

Niu, N., Savolainen, J., Bhowmik, T., Mahmoud, A., and Reddivari, S. (2012). A framework for examining topical locality in object-oriented software. In *Proceedings of the 2012 IEEE 36th Annual Computer Software and Applications Conference*, pages 219–224.

Oliveto, R., Gethers, M., Poshyvanyk, D., and De Lucia, A. (2010). On the equivalence of information retrieval methods for automated traceability link recovery.

In *Proceedings of the 18th International Conference on Program Comprehension*, pages 68–71.

Oliveto, R., Gethers, M., Bavota, G., Poshyvanyk, D., and De Lucia, A. (2011). Identifying method friendships to remove the feature envy bad smell. In *Proceeding of the 33rd International Conference on Software Engineering (NIER Track)*, pages 820–823.

Ossher, J., Bajracharya, S., Linstead, E., Baldi, P., and Lopes, C. (2009). Sourcererdb: An aggregated repository of statically analyzed and cross-linked open source java projects. In *Proceedings of the 6th Working Conference on Mining Software Repositories*, pages 183–186.

Pagano, D. and Maalej, W. (2013). How do open source communities blog? *Empirical Softw. Engg.*, **18**(6), 1090–1124.

Panichella, A., Dit, B., Oliveto, R., Di Penta, M., Poshyvanyk, D., and De Lucia, A. (2013). How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 522–531.

Parizy, M., Takayama, K., and Kanazawa, Y. (2014). Software defect prediction for lsi designs. In *Proceedings of the 2014 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 565–568.

Paul, M. (2009). *Cross-Collection Topic Models: Automatically Comparing and Contrasting Text*. Master's thesis, University of Illinois at Urbana-Champaign, Urbana, IL.

Petersen, K., Feldt, R., Mujtaba, S., and Mattsson, M. (2008). Systematic mapping studies in software engineering. In *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, EASE'08, pages 68–77.

Phan, X.-H., Nguyen, L.-M., and Horiguchi, S. (2008a). Learning to classify short and sparse text & web with hidden topics from large-scale data collections. In *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, pages 91–100.

Phan, X. H., Nguyen, L. M., and Horiguchi, S. (2008b). Learning to classify short and sparse text & web with hidden topics from large-scale data collections. In *Proceeding of the 17th International Conference on World Wide Web*, pages 91–100.

Pingclasai, N., Hata, H., and Matsumoto, K.-i. (2013). Classifying bug reports to bugs and other requests using topic modeling. In *Proceedings of the 2013 20th Asia-Pacific Software Engineering Conference (APSEC)*, APSEC '13, pages 13–18.

Porteous, I., Newman, D., Ihler, A., Asuncion, A., Smyth, P., and Welling, M. (2008). Fast collapsed Gibbs sampling for latent Dirichlet allocation. In *Proceeding of the 14th International Conference on Knowledge Discovery and Data Mining*, pages 569–577.

Porter, M. (1980). An algorithm for suffix stripping. *Program*, **14**, 130.

Poshyvanyk, D. and Grechanik, M. (2009). Creating and evolving software by searching, selecting and synthesizing relevant source code. In *Proceedings of the 31st International Conference on Software Engineering*, pages 283–286.

Poshyvanyk, D. and Marcus, A. (2007). Combining formal concept analysis with information retrieval for concept location in source code. In *Proceedings of the 15th International Conference on Program Comprehension*, pages 37–48.

Poshyvanyk, D., Marcus, A., Rajlich, V., *et al.* (2006). Combining probabilistic ranking and Latent Semantic Indexing for feature identification. In *Proceedings of the 14th International Conference on Program Comprehension*, pages 137–148.

Poshyvanyk, D., Gueheneuc, Y., Marcus, A., Antoniol, G., and Rajlich, V. (2007). Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval. *IEEE Transactions on Software Engineering*, **33**(6), 420–432.

Poshyvanyk, D., Gethers, M., and Marcus, A. (2013). Concept location using formal concept analysis and information retrieval. *ACM Trans. Softw. Eng. Methodol.*, **21**(4), 23:1–23:34.

Qusef, A., Bavota, G., Oliveto, R., Lucia, A. D., and Binkley, D. (2013). Evaluating test-to-code traceability recovery methods through controlled experiments. *Journal of Software: Evolution and Process*, **25**(11), 1167–1191.

Rahman, F., Bird, C., and Devanbu, P. T. (2012). Clones: what is that smell? *Empirical Software Engineering*, **17**(4-5), 503–530.

Raja, U. (2012). All complaints are not created equal: text analysis of open source software defect reports. *Empirical Softw. Engg.*

Rajlich, V. and Wilde, N. (2002). The role of concepts in program comprehension. In *Proceedings of the 10th International Workshop on Program Comprehension*, pages 271–278.

Ramage, D., Hall, D., Nallapati, R., and Manning, C. D. (2009a). Labeled LDA: a supervised topic model for credit attribution in multi-labeled corpora. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1*, pages 248–256.

Ramage, D., Rosen, E., Chuang, J., Manning, C. D., and McFarland, D. A. (2009b). Topic modeling for the social sciences. In *NIPS 2009 Workshop on Applications for Topic Models: Text and Beyond*.

Rao, S. and Kak, A. (2011). Retrieval from software libraries for bug localization: a comparative study of generic and composite text models. In *Proceeding of the 8th Working Conference on Mining Software Repositories*, pages 43–52.

Revelle, M. and Poshyvanyk, D. (2009). An exploratory study on assessing feature location techniques. In *Proceedings of the 17th International Conference on Program Comprehension*, pages 218–222.

Revelle, M., Dit, B., and Poshyvanyk, D. (2010). Using data fusion and web mining to support feature location in software. In *Proceedings of the 18th International Conference on Program Comprehension*, pages 14–23.

Risi, M., Scanniello, G., and Tortora, G. (2010). Architecture recovery using latent semantic indexing and k-means: An empirical evaluation. In *Proceedings of the 2010 8th IEEE International Conference on Software Engineering and Formal Methods (SEFM)*, pages 103–112.

Rosen-Zvi, M., Griffiths, T., Steyvers, M., and Smyth, P. (2004). The author-topic model for authors and documents. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 487–494.

Roy, C. K., Cordy, J. R., and Koschke, R. (2009). Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Science of Computer Programming*, **74**(7), 470–495.

Saha, R., Lease, M., Khurshid, S., and Perry, D. (2013). Improving bug localization using structured information retrieval. In *Proceedings of the 2013 IEEE/ACM*

28th International Conference on Automated Software Engineering, pages 345–355.

Salton, G. and McGill, M. J. (1983). Introduction to modern information retrieval. *New York*.

Salton, G., Wong, A., and Yang, C. S. (1975). A vector space model for automatic indexing. *Communications of the ACM*, **18**(11), 620.

Savage, T., Dit, B., Gethers, M., and Poshyvanyk, D. (2010). TopicXP: Exploring topics in source code using latent Dirichlet allocation. In *Proceedings of the 26th International Conference on Software Maintenance*, pages 1–6.

Shang, W., Jiang, Z. M., Adams, B., Hassan, A. E., Godfrey, M. W., Nasser, M., and Flora, P. (2013). An exploratory study of the evolution of communicated information about the execution of large software systems. *Journal of Software: Evolution and Process*, **26**(1), 3–26.

Sharafl, Z., Gueheneuc, Y.-G., Ali, N., and Antoniol, G. (2012). An empirical study on requirements traceability using eye-tracking. In *Proceedings of the 2012 IEEE International Conference on Software Maintenance*, ICSM '12, pages 191–200.

Shihab, E., Jiang, Z. M., Ibrahim, W. M., Adams, B., and Hassan, A. E. (2010). Understanding the impact of code and process metrics on post-release defects: a case study on the eclipse project. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement*.

Somasundaram, K. and Murphy, G. C. (2012). Automatic categorization of bug reports using latent dirichlet allocation. In *Proceedings of the 5th India Software Engineering Conference*, ISEC '12, pages 125–130.

Steyvers, M. and Griffiths, T. (2007). Probabilistic topic models. In *Latent Semantic Analysis: A Road to Meaning*. Laurence Erlbaum.

Tairas, R. and Gray, J. (2009). An information retrieval process to aid in the analysis of code clones. *Empirical Softw. Engg.*, **14**(1), 33–56.

Tang, J., Meng, Z., Nguyen, X., Mei, Q., and Zhang, M. (2014). Understanding the limiting factors of topic modeling via posterior contraction analysis. In T. Jebara and E. P. Xing, editors, *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 190–198.

Thomas, S., Nagappan, M., Blostein, D., and Hassan, A. (2013). The impact of classifier configuration and classifier combination on bug localization. *IEEE Transactions on Software Engineering*, **39**(10), 1427–1443.

Thomas, S. W., Adams, B., Hassan, A. E., and Blostein, D. (2010). Validating the use of topic models for software evolution. In *Proceedings of the 10th International Working Conference on Source Code Analysis and Manipulation*, pages 55–64.

Thomas, S. W., Hemmati, H., Hassan, A. E., and Blostein, D. (2014). Static test case prioritization using topic models. *Empirical Software Engineering*.

Tian, K., Revelle, M., and Poshyvanyk, D. (2009). Using latent Dirichlet allocation for automatic categorization of software. In *Proceedings of the 6th International Working Conference on Mining Software Repositories*, pages 163–166.

Tichy, W. (2010). An interview with Prof. Andreas Zeller: Mining your way to software reliability. *Ubiquity*, **2010**.

Ujhazi, B., Ferenc, R., Poshyvanyk, D., and Gyimothy, T. (2010). New conceptual coupling and cohesion metrics for object-oriented systems. In *Proceedings of the 10th International Working Conference on Source Code Analysis and Manipulation*, pages 33–42.

Van der Spek, P., Klusener, S., and Van de Laar, P. (2008). Towards recovering architectural concepts using Latent Semantic Indexing. In *Proceedings of the 12th European Conference on Software Maintenance and Reengineering*, pages 253–257.

Wall, M., Rechtsteiner, A., and Rocha, L. (2003). Singular value decomposition and principal component analysis. pages 91–109.

Wallach, H., Mimno, D., and McCallum, A. (2009a). Rethinking LDA: why priors matter. *Proceedings of NIPS-09, Vancouver, BC*.

Wallach, H. M., Murray, I., Salakhutdinov, R., and Mimno, D. (2009b). Evaluation methods for topic models. In *Proceedings of the 26th International Conference on Machine Learning*, pages 1105–1112.

Wang, C., Thiesson, B., Meek, C., and Blei, D. (2009). Markov topic models. In *The Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 583–590.

Wang, S., Lo, D., Xing, Z., and Jiang, L. (2011). Concern localization using information retrieval: An empirical study on linux kernel. In *Proceedings of the 2011 18th Working Conference on Reverse Engineering*, WCRE '11, pages 92–96.

Wang, X. and McCallum, A. (2006). Topics over time: a non-markov continuous-time model of topical trends. In *Proceedings of the 12th international conference on Knowledge discovery and data mining*, pages 424–433. ACM.

Wu, C., Chang, E., and Aitken, A. (2008). An empirical approach for semantic web services discovery. In *Proceedings of the 19th Australian Conference on Software Engineering*, pages 412–421.

Xia, X., Lo, D., Wang, X., and Zhou, B. (2013). Accurate developer recommendation for bug resolution. In *Proceedings of the 2013 20th Working Conference on Reverse Engineering*, pages 72–81.

Xie, B., Li, M., Jin, J., Zhao, J., and Zou, Y. (2013). Mining cohesive domain topics from source code. In *Proceedings of the International Conference on Software Reuse*, pages 239–254.

Xue, Y., Xing, Z., and Jarzabek, S. (2012). Feature location in a collection of product variants. In *Proceedings of the 2012 19th Working Conference on Reverse Engineering*, pages 145–154.

Yan, X., Guo, J., Lan, Y., and Cheng, X. (2013). A biterm topic model for short texts. In *Proceedings of the 22nd International Conference on World Wide Web*, WWW '13, pages 1445–1456.

Yang, G., Zhang, T., and Lee, B. (2014). Towards semi-automatic bug triage and severity prediction based on topic model and multi-feature of bug reports. In *Proceedings of the 2014 IEEE 38th Annual Computer Software and Applications Conference*, COMPSAC '14, pages 97–106.

Yu, S. (2012). Retrieving software maintenance history with topic models. In *Proceedings of the 2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pages 621–624.

Zawawy, H., Kontogiannis, K., and Mylopoulos, J. (2010). Log filtering and interpretation for root cause analysis. In *Proceedings of the 26th International Conference on Software Maintenance*, pages 1–5.

Zhai, C. X. (2008). Statistical language models for information retrieval. *Synthesis Lectures on Human Language Technologies*, **1**(1), 1–141.

Zhou, J., Zhang, H., and Lo, D. (2012). Where should the bugs be fixed? - more accurate information retrieval-based bug localization based on bug reports. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 14–24.

Zimmermann, T., Weisgerber, P., Diehl, S., and Zeller, A. (2005). Mining version histories to guide software changes. *IEEE Transactions on Software Engineering*, pages 429–445.

Zou, C. and Hou, D. (2014). Lda analyzer: A tool for exploring topic models. In *Proceedings of the 2014 International Conference on Software Maintenance and Evolution (ICSME)*, pages 593–596.

Appendix

A Article Selection Process

In this paper, we are interested in locating articles that use topic modeling techniques to solve SE tasks. We focus our attention on articles written between December 1999 and December 2014, more than a full decade of research results. We consider in our search a wide range of journals and conferences in an effort to cover as many relevant articles as possible.

To select our list of articles, we use first compile a list of highly relevant venues to search. Then, we perform a series of keyword searches at each venue, producing a list of candidate articles. For each of the candidate articles, we read the abstract (and, in some cases, the introduction) of the article to determine if the article is indeed relevant to our interests. This yields an *initial set* of related articles. For each of the articles in the initial set, we consider the citations that are contained in the article for additional relevant articles. Then, we reach our *final set* of articles.

A.1 Considered Venues

Table A.1 lists the journals and conference venues that we included in our initial search for articles.

A.2 Keyword Searches and Filtering

We collected the initial set of articles by performing keyword searches at the publisher websites for each of our considered venues. We also searched using aggregate search engines, such as the ACM Digital Library and IEEE Xplore. The keywords and search queries that we use are listed below.

IEEE Xplore

```
("topic models" OR "topic model"
OR "lsi" OR "lda" OR "plsi"
OR "latent dirichlet allocation" OR "latent semantic")
AND
(  "Publication Title":"Source Code Analysis and Manipulation"
OR "Publication Title":"Software Engineering, IEEE Transactions on"
OR "Publication Title":"Reverse Engineering"
OR "Publication Title":"Software Maintenance"
OR "Publication Title":"Software Engineering"
OR "Publication Title":"Program Comprehension"
OR "Publication Title":"Mining Software Repositories")
```

Table A1 The fifteen venues that we considered in our initial article selection process.

Type	Acronym	Description
Journal	TSE	IEEE Transactions on Software Engineering
	TOSEM	ACM Transactions on Software Engineering & Methodology
	EMSE	Empirical Software Engineering
	JSS	Journal of Systems and Software
	JSME	Journal of Software Maintenance and Evolution
	SP&E	Software – Practice & Experience
Conference	ICSE	International Conference on Software Engineering
	ESEC/FSE	European Software Engineering Conference / Symposium on the Foundations of Software Engineering
	FASE	International Conference on Fundamental Approaches to Software Engineering
	ASE	International Conference on Automated Software Engineering
	ICSM	International Conference on Software Maintenance
	WCRE	Working Conference on Reverse Engineering
	IWPC/ICPC	International Workshop/Conference on Program Comprehension
	SCAM	International Workshop/Working Conference on Source Code Analysis and Manipulation
	MSR	International Workshop/Working Conference on Mining Software Repositories

Software Practice and Experience

```
lsi or lda or "topic model" or "topic models" or
"latent dirichlet allocation" or "latent semantic"
AND publication title="Software Practice and Experience"
```

Journal of Software Maintenance and Evolution

```
lsi or lda or "topic model" or "topic models" or
"latent dirichlet allocation" or "latent semantic"
AND publication title="Software Maintenance and Evolution"
```

Empirical Software Engineering

```
lsi or lda or "topic model" or "topic models" or
latent dirichlet allocation" or "latent semantic"
```

In general, we found that keyword searches resulted in many irrelevant results. We manually filtered the search results by reading the article's abstract (and sometimes introduction) to determine if the article solved an SE task by employing one or more topic modeling technique. The articles that were determined to be relevant were added to our initial set.

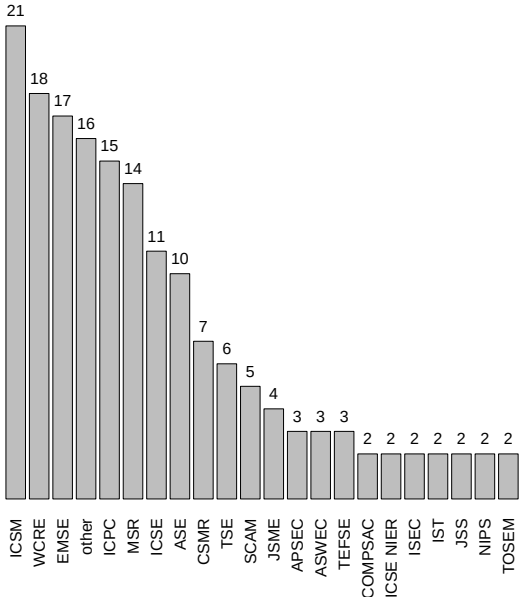
A.3 Reference Checking

For each article in the initial set, we followed its citations to obtain another list of potentially relevant articles. Again, we filtered this list by reading the abstract and introduction. The articles that were determined to relevant were added to our final set of articles.

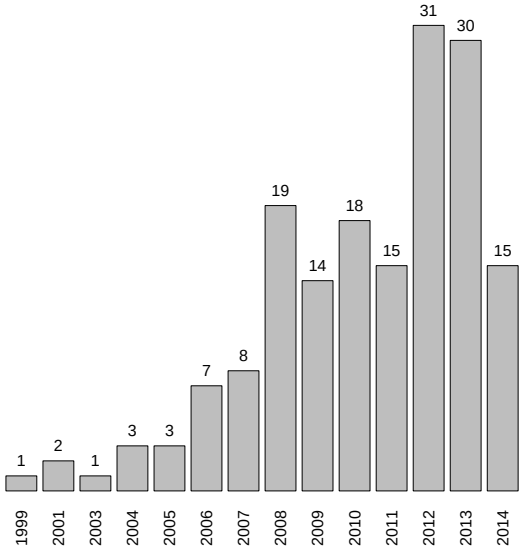
A.4 Article Selection Results

We finally arrive at 167 articles published between 1999 and 2014. Figure A1 shows the distribution of venues and years for the articles.

B Article characterization results for facets 1–6.



(a) Venues of articles



(b) Years of articles

Fig. A1 Article selection results.

Table B1 Article characterization results of facets 1–4. The attributes are described in Table 2.

	IR model			Task									Repository					Evaluation			
	LSI	LDA	other	doc. clustering	concept loc.	metrics	trend/evolution	traceability	bug pred./debug	org./search coll.	other	source code	email	req./design	logs	bug reports	statistical	task specific	manual	user study
Ahsan *et al.* (2009)	o	.	.	.	.	.	.	.	.	.	o	.	.	.	.	o	.	o	.	.
Alhindawi *et al.* (2013a)	o	.	.	.	o	.	.	.	.	.	.	o	.	.	.	.	.	o	.	.
Ali *et al.* (2012)	o	.	.	.	o	.	.	.	.	.	.	o	.	.	.	o	.	o	.	.
Alipour *et al.* (2013)	.	o	o	.	.	.	.	.	.	.	o	.	.	.	.	o	.	o	.	.
Andrzejewski *et al.* (2007)	.	o	.	.	.	.	.	.	o	.	.	.	.	.	o	.	.	o	.	.
Antoniol *et al.* (2008)	o	.	.	.	.	.	.	o	.	.	.	o	.	o	.	.	.	.	o	.
Asuncion *et al.* (2010)	.	o	.	.	.	.	.	o	.	.	.	o	o	o	.	o	.	o	.	.
Bajracharya and Lopes (2009)	.	o	.	.	.	.	.	.	.	.	o	.	.	.	o	.	.	.	.	.
Bajracharya and Lopes (2010)	.	o	.	.	.	.	.	.	.	.	o	.	.	.	o	.	.	.	.	.
Baldi *et al.* (2008)	.	o	.	.	o	.	.	.	.	o	.	o	.	.	.	.	.	.	.	.
Bassett and Kraft (2013)	.	o	.	.	o	.	.	.	.	.	.	o	.	.	.	.	.	o	.	.
Bavota *et al.* (2010)	o	.	.	.	.	o	.	.	.	.	.	o	.	.	.	.	.	o	.	.
Bavota *et al.* (2013)	o	.	.	.	.	.	.	o	.	.	.	o	.	o	.	.	.	o	.	.
Beard *et al.* (2011)	o	o	.	.	.	.	.	o	.	.	.	o	.	.	.	.	.	o	.	.
Binkley *et al.* (2012)	o	.	.	.	.	.	.	o	.	.	.	o	.	.	.	.	.	o	.	.
Bose and Suresh (2008)	o	.	.	o	.	.	.	.	o	.	.	.	.	.	o	.	.	o	.	.
Campbell *et al.* (2013)	.	o	.	.	.	.	.	o	.	.	.	.	.	o	.	.	.	o	.	.
Capobianco *et al.* (2009)	o	.	.	.	.	.	.	o	.	.	.	o	.	o	.	.	.	o	.	.
Chen *et al.* (2012)	.	o	.	.	.	o	.	.	o	.	.	o	.	.	.	.	.	o	o	.
Cleary *et al.* (2008)	o	.	.	.	o	.	.	.	.	.	.	o	.	.	.	.	.	o	.	.
Corley *et al.* (2012)	.	o	.	o	.	.	.	.	.	.	o	o	.	.	.	.	.	o	.	.
Dasgupta *et al.* (2013)	o	o	o	.	.	.	.	o	.	.	.	o	.	.	.	.	.	o	.	.
De Lucia *et al.* (2011)	o	.	.	.	.	.	.	o	.	.	.	o	.	o	.	.	.	o	.	.
de Boer and van Vliet (2008)	o	.	.	.	.	.	.	o	.	.	o	.	.	o	.	.	.	.	.	.
De Lucia *et al.* (2004)	o	.	.	.	.	.	.	o	.	.	.	o	.	o	.	.	.	o	.	.
De Lucia *et al.* (2006)	o	.	.	.	.	.	.	o	.	.	.	o	.	o	.	.	.	o	.	.
De Lucia *et al.* (2007)	o	.	.	.	.	.	.	o	.	.	.	o	.	o	.	.	.	o	.	.
Dit *et al.* (2008)	o	.	.	.	.	.	.	.	.	.	o	.	.	.	.	o	.	o	o	.
Dit *et al.* (2013a)	.	o	.	.	.	.	.	.	.	.	o	o	.	o	.	.	.	o	.	.
Eddy *et al.* (2013)	.	.	o	.	.	.	.	.	.	.	o	o	.	.	.	.	.	.	.	o
Eyal-Salman *et al.* (2013)	o	.	.	.	.	.	.	o	.	.	.	o	.	o	.	.	.	o	.	.
Gall *et al.* (2008)	o	.	.	.	.	o	.	.	.	.	.	o	.	o	.	.	.	.	.	.
Gethers and Poshyvanyk (2010)	.	o	.	.	.	o	.	.	.	.	.	o	.	.	.	.	.	o	.	.
Gethers *et al.* (2011b)	o	.	o	.	o	.	.	o	.	.	.	o	.	.	.	.	.	o	.	.
Gethers *et al.* (2011d)	o	.	o	.	.	.	.	o	.	.	.	o	.	o	.	.	.	o	.	.
Gethers *et al.* (2012)	o	.	.	.	.	.	.	o	.	.	.	o	.	.	.	o	.	o	.	.
Gethers *et al.* (2011a)	o	.	.	.	.	.	.	o	.	.	.	o	.	.	.	o	.	o	.	.
Grant *et al.* (2008)	.	.	o	.	o	.	.	.	.	.	.	o	.	.	.	.	.	.	.	.
Grant and Cordy (2009)	.	.	o	.	.	.	.	.	.	.	o	o	.	.	.	.	.	.	.	.
Grant and Cordy (2010)	.	o	.	.	.	.	.	.	.	.	o	o	.	.	.	.	.	o	.	.

continued on next page

continued from previous page

	IR model			Task								Repository					Evaluation			
	LSI	LDA	other	doc. clustering	concept loc.	metrics	trend/evolution	traceability	bug pred./debug	org./search coll.	other	source code	email	req./design	logs	bug reports	statistical	task specific	manual	user study
Grant *et al.* (2011b)	o	o	.	.	o	.	.	.	.	.	.	o	.	.	o	.	.	o	.	.
Grant *et al.* (2011a)	.	o	.	o	.	.	.	.	.	.	.	o	.	.	.	.	.	o	.	.
Han *et al.* (2012)	.	o	o	.	.	.	o	.	.	.	o	.	.	.	.	o	.	o	o	.
Hayes *et al.* (2006)	o	.	.	.	.	.	.	o	.	.	.	.	.	o	.	.	.	o	.	.
Hindle *et al.* (2009)	.	o	.	.	.	.	o	.	.	.	.	.	.	.	o	.	.	.	.	.
Hindle *et al.* (2010)	.	o	.	.	.	.	o	.	.	.	.	.	.	.	o	.	.	.	.	.
Hindle *et al.* (2011)	.	o	.	.	.	.	.	.	.	o	.	.	.	.	o	.	.	o	o	.
Hindle *et al.* (2012c)	.	o	.	.	.	.	o	.	.	.	o	.	.	o	o	.	.	.	.	o
Hu and Wong (2013)	.	.	o	.	.	o	.	.	o	.	.	o	.	.	o	.	.	o	.	.
Iacob and Harrison (2013)	.	o	.	.	.	.	o	.	.	.	o	.	.	.	.	.	.	o	.	.
Islam *et al.* (2012b)	o	.	.	.	.	.	.	o	.	.	o	o	.	o	.	.	.	o	.	.
Jiang *et al.* (2008)	o	.	.	.	.	.	.	o	.	.	.	o	.	o	.	.	.	o	.	.
Kagdi *et al.* (2010)	o	.	.	.	.	o	.	.	.	.	.	o	.	.	.	.	.	o	.	.
Kagdi *et al.* (2012a)	o	.	.	.	.	.	.	.	.	.	o	o	.	.	.	.	.	o	.	.
Kaushik *et al.* (2011)	o	o	.	.	.	.	.	o	.	.	.	o	.	.	.	o	.	o	o	.
Kaushik and Tahvildari (2012)	o	o	.	o	.	.	.	.	.	.	.	.	.	.	.	o	.	o	.	.
Kawaguchi *et al.* (2006)	o	.	.	.	.	.	.	.	.	o	.	o	.	.	.	.	.	.	.	.
Kouters *et al.* (2012)	o	.	.	.	.	.	.	.	.	.	o	.	.	.	o	.	.	o	.	.
Kuhn *et al.* (2005)	o	.	.	o	.	.	.	.	.	.	.	o	.	.	.	.	.	.	.	.
Kuhn *et al.* (2007)	o	.	.	o	.	.	.	.	.	.	.	o	.	.	.	.	.	.	.	.
Kuhn *et al.* (2008)	o	.	.	o	.	.	o	.	.	.	.	o	.	.	.	.	.	o	.	.
Kuhn *et al.* (2010)	o	.	.	o	.	.	o	.	.	.	.	o	.	.	.	.	.	.	.	.
Lin *et al.* (2006)	o	.	.	o	.	.	.	.	.	.	.	o	.	.	.	.	.	o	o	.
Linstead *et al.* (2007b)	.	o	.	.	o	o	.	.	.	o	.	o	.	.	.	.	.	.	.	.
Linstead *et al.* (2007a)	.	o	.	.	o	.	.	.	.	.	o	o	.	.	.	.	.	.	.	.
Linstead *et al.* (2008b)	.	o	.	o	o	o	.	.	.	o	o	o	.	.	.	.	.	.	.	.
Linstead *et al.* (2008c)	.	o	.	o	o	o	.	.	.	.	o	o	.	.	.	.	.	.	.	.
Linstead *et al.* (2008a)	.	o	.	.	o	.	o	.	.	.	.	o	.	.	.	.	.	.	.	.
Linstead and Baldi (2009)	.	o	.	.	.	o	.	.	.	.	.	.	.	.	.	o	.	.	.	.
Linstead *et al.* (2009)	.	o	.	.	o	.	o	.	.	.	o	o	.	.	.	.	.	.	.	.
Liu *et al.* (2009)	.	o	.	.	.	o	.	.	o	.	.	o	.	.	.	.	.	o	.	.
Lohar *et al.* (2013)	o	.	.	.	.	.	.	o	.	.	.	o	.	o	.	.	.	o	.	.
Lormans and Van Deursen (2006)	o	.	.	.	.	.	.	o	.	.	.	o	.	o	.	.	.	o	.	.
Lormans *et al.* (2006)	o	.	.	.	.	.	.	o	.	.	.	.	.	o	.	.	.	o	.	.
Lormans (2007)	o	.	.	.	.	.	o	o	.	.	.	.	.	o	.	.	.	o	.	.
De Lucia *et al.* (2012)	o	o	.	.	.	.	.	.	.	.	o	o	.	.	.	.	.	o	o	o
De Lucia *et al.* (2014)	o	o	.	.	.	.	.	.	.	.	o	o	.	.	.	.	.	o	o	o
Lukins *et al.* (2008)	.	o	.	.	.	.	.	o	.	.	.	o	.	.	.	o	.	o	.	.
Lukins *et al.* (2010)	.	o	.	.	.	.	.	o	.	.	.	o	.	.	.	o	.	o	.	.
Maletic and Valluri (1999)	o	.	.	o	.	.	.	.	.	.	.	o	.	.	.	.	.	.	.	.

continued on next page

continued from previous page

	IR model			Task								Repository					Evaluation			
	LSI	LDA	other	doc. clustering	concept loc.	metrics	trend/evolution	traceability	bug pred./debug	org./search coll.	other	source code	email	req./design	logs	bug reports	statistical	task specific	manual	user study
Maletic and Marcus (2001)	o	.	.	o	.	.	.	.	.	.	.	o	.	.	.	.	.	.	o	.
Marcus and Maletic (2001)	o	.	.	.	.	.	.	.	.	.	o	o	.	.	.	.	.	.	o	.
Marcus and Maletic (2003)	o	.	.	.	.	.	.	o	.	.	.	o	.	o	.	.	.	o	.	.
Marcus *et al.* (2004)	o	.	.	.	o	.	.	.	.	.	.	o	.	.	.	.	.	o	o	.
Marcus (2004)	o	.	.	o	o	o	.	o	.	.	.	o	.	o	.	.	.	.	.	.
Marcus *et al.* (2005)	o	.	.	.	o	.	.	.	.	.	.	o	.	.	.	.	.	.	o	.
Marcus *et al.* (2008)	o	.	.	.	.	o	.	.	o	.	.	o	.	.	.	.	.	o	.	.
Maskeri *et al.* (2008)	.	o	.	.	o	.	.	.	.	.	.	o	.	.	.	.	o	.	.	.
McMillan *et al.* (2009)	o	.	.	.	.	.	.	o	.	.	.	o	.	o	.	.	.	o	o	o
Misra *et al.* (2012)	o	.	.	o	.	.	.	.	.	.	.	o	.	.	.	.	.	o	.	.
Moritz *et al.* (2013)	.	.	o	.	.	.	.	.	.	.	o	o	.	.	.	.	.	o	.	.
Naguib *et al.* (2013)	.	o	.	o	.	.	.	.	.	.	o	.	.	.	.	o	.	o	.	.
Neuhaus and Zimmermann (2010)	.	o	.	.	.	.	o	.	.	.	.	.	.	.	.	o	.	.	.	.
Nguyen *et al.* (2011b)	.	o	.	.	.	.	.	.	o	.	.	o	.	.	.	.	.	o	o	.
Nguyen *et al.* (2011a)	.	.	o	.	o	.	.	.	.	.	o	o	.	.	.	o	.	o	.	.
Nguyen *et al.* (2012)	.	o	.	.	.	.	.	.	.	.	o	.	.	.	.	o	.	o	.	.
Nie and Zhang (2012)	.	o	.	.	o	.	.	.	.	.	.	o	.	.	.	.	.	o	.	.
Niu *et al.* (2012)	.	.	o	.	o	.	.	.	.	.	.	o	.	.	.	.	.	o	.	.
Oliveto *et al.* (2010)	o	o	.	.	.	.	.	o	.	.	.	o	.	.	.	.	.	o	.	.
Oliveto *et al.* (2011)	.	.	o	.	.	.	.	.	o	.	.	o	.	.	.	.	.	o	.	.
Ossher *et al.* (2009)	.	o	.	.	o	.	o	.	.	o	.	o	.	.	.	.	.	.	.	.
Panichella *et al.* (2013)	.	o	.	.	.	.	.	.	.	.	o	o	.	.	.	.	o	o	.	.
Poshyvanyk *et al.* (2006)	o	.	.	.	o	.	.	.	.	.	.	o	.	.	.	.	.	o	.	.
Poshyvanyk and Marcus (2007)	o	.	.	.	o	.	.	.	.	.	.	o	.	.	.	.	.	o	.	.
Poshyvanyk *et al.* (2007)	o	.	.	.	o	.	.	.	.	.	.	o	.	.	.	.	.	o	.	.
Poshyvanyk and Grechanik (2009)	o	.	.	.	o	.	.	.	.	.	o	o	.	.	.	.	.	.	.	.
Poshyvanyk *et al.* (2013)	o	.	.	.	o	.	.	.	.	.	.	o	.	.	.	.	.	o	.	.
Qusef *et al.* (2013)	o	.	.	.	.	.	.	o	.	.	.	o	.	.	.	.	.	o	.	.
Revelle and Poshyvanyk (2009)	o	.	.	.	o	.	.	.	.	.	.	o	.	.	.	.	.	o	.	.
Revelle *et al.* (2010)	o	.	.	.	o	.	.	.	.	.	.	o	.	.	.	.	.	o	.	.
Saha *et al.* (2013)	o	.	.	.	o	.	.	.	.	.	o	o	.	.	.	o	.	.	.	.
Savage *et al.* (2010)	.	o	.	.	o	.	.	.	.	.	.	o	.	.	.	.	.	.	.	.
Shang *et al.* (2013)	.	o	.	.	.	.	.	.	.	.	.	.	.	.	o	.	.	.	o	.
Sharafl *et al.* (2012)	o	o	.	.	.	.	.	o	.	.	.	o	.	o	.	.	.	o	o	.
Thomas *et al.* (2010)	.	o	.	.	.	.	o	.	.	.	.	o	.	.	.	.	.	.	o	.
Thomas *et al.* (2013)	o	o	o	.	.	.	.	o	.	.	.	o	.	.	.	o	.	o	.	.
Tian *et al.* (2009)	.	o	.	.	.	.	.	.	.	o	.	o	.	.	.	.	.	.	.	.
Ujhazi *et al.* (2010)	o	.	.	.	.	o	.	.	o	.	.	o	.	.	.	.	.	o	.	.
Van der Spek *et al.* (2008)	o	.	.	.	o	.	.	.	.	.	.	o	.	.	.	.	.	.	o	.
Wang *et al.* (2011)	o	o	o	.	o	.	.	.	.	.	.	o	.	.	.	.	.	o	.	.

continued on next page

continued from previous page

	IR model			Task								Repository					Evaluation			
	LSI	LDA	other	doc. clustering	concept loc.	metrics	trend/evolution	traceability	bug pred./debug	org./search coll.	other	source code	email	req./design	logs	bug reports	statistical	task specific	manual	user study
Wu *et al.* (2008)	o	.	.	.	.	.	.	.	.	.	o	.	.	.	o	.	.	.	.	.
Xia *et al.* (2013)	.	o	.	.	.	.	.	.	.	.	o	.	.	.	.	o	.	o	.	.
Xie *et al.* (2013)	.	o	.	.	.	.	.	.	.	.	o	o	.	.	.	.	.	.	.	.
Xue *et al.* (2012)	o	.	.	.	.	.	.	o	.	.	.	o	.	.	.	.	.	o	.	.
Zawawy *et al.* (2010)	o	.	.	.	.	.	.	.	o	.	.	.	.	.	o	.	.	o	.	.
Zhou *et al.* (2012)	o	o	o	.	.	.	.	.	o	.	.	o	.	.	.	o	.	o	.	.
Medini (2011)	.	o	.	.	o	.	.	.	.	.	.	.	.	.	o	.	.	o	.	.
Zou and Hou (2014)	.	o	.	.	.	.	.	.	.	.	o	.	.	.	.	.	.	.	.	.
Limsettho *et al.* (2014)	.	o	o	.	.	.	.	.	.	.	o	.	.	.	.	o	.	o	.	.
Grant and Cordy (2014)	.	o	.	.	o	.	.	.	.	.	.	o	.	.	o	.	.	o	.	.
Yu (2012)	.	o	.	.	.	.	o	.	.	.	.	.	.	.	.	o	.	o	o	.
Thomas *et al.* (2014)	.	o	.	.	.	.	.	.	.	.	o	o	.	.	.	.	.	o	.	.
Grant *et al.* (2012)	.	o	.	.	.	.	o	o	.	.	.	o	.	.	o	.	.	o	.	.
Bavota *et al.* (2014)	.	.	o	.	.	.	.	o	.	.	.	o	.	.	.	.	.	o	.	o
Alhindawi *et al.* (2013b)	o	.	.	.	.	.	.	o	.	.	.	o	.	o	.	.	.	o	.	.
Parizy *et al.* (2014)	o	.	.	.	.	.	.	.	.	.	o	o	.	.	o	.	.	o	.	.
Islam *et al.* (2012a)	o	.	.	.	.	.	.	.	.	.	o	o	.	o	.	.	.	o	.	.
Le *et al.* (2013)	.	o	.	.	o	.	.	.	.	.	.	o	.	.	.	o	.	o	.	.
Misra and Das (2013)	o	.	.	.	.	.	.	.	.	.	o	.	.	o	.	.	.	o	.	.
Asadi *et al.* (2010b)	o	.	.	.	o	.	.	.	.	.	.	o	.	.	o	.	.	o	o	.
Dit *et al.* (2013b)	o	.	.	.	o	.	.	.	.	.	.	o	.	o	.	.	.	o	.	.
Tairas and Gray (2009)	o	.	.	o	.	.	.	.	.	.	.	o	.	.	.	.	.	o	.	.
Lormans *et al.* (2008)	o	.	.	.	.	.	.	o	.	.	.	o	.	o	.	.	.	o	.	.
Ali *et al.* (2014)	o	o	.	.	.	.	.	o	.	.	.	o	.	o	.	.	.	o	.	.
Kagdi *et al.* (2012b)	o	.	.	.	.	o	.	.	.	.	.	o	.	.	.	.	.	o	.	.
Bavota *et al.* (2012)	o	.	.	.	.	.	.	.	.	.	o	o	.	.	.	.	.	o	.	.
Raja (2012)	.	.	.	o	.	.	.	.	.	.	.	.	.	.	.	o	.	o	.	.
Barua *et al.* (2012)	.	o	.	.	.	.	.	.	.	o	.	.	.	.	.	.	.	o	o	.
Hindle *et al.* (2014)	.	o	.	.	.	.	o	o	.	.	o	.	.	o	o	.	.	.	.	o
Hindle *et al.* (2012a)	.	o	.	.	.	.	.	.	.	.	o	.	.	.	o	.	.	o	o	.
Pagano and Maalej (2013)	.	o	.	.	.	.	.	.	.	o	.	.	.	.	.	.	.	o	o	.
Biggers *et al.* (2014)	.	o	.	.	o	.	.	.	.	.	o	o	.	.	.	.	.	o	.	.
Canfora *et al.* (2014)	.	o	.	.	.	.	o	.	.	.	.	.	.	.	o	.	.	o	o	.
Gorla *et al.* (2014)	.	o	.	.	.	.	.	.	.	o	.	.	.	.	.	.	.	o	.	.
Linares-Vásquez *et al.* (2013)	.	o	.	.	.	.	o	.	.	o	.	.	.	.	.	.	.	o	o	.
Allamanis and Sutton (2013)	.	o	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	o	o	.
Bajaj *et al.* (2014)	.	o	.	.	.	.	o	.	.	.	.	.	.	.	.	.	.	o	o	.
Brickey *et al.* (2012)	o	.	.	o	.	.	.	.	.	.	.	.	.	.	o	.	.	o	o	.
Pingclasai *et al.* (2013)	.	o	.	o	.	.	.	.	.	.	o	.	.	.	.	o	.	o	.	.
Galvis Carreño and Winbladh (2013)	.	.	o	o	.	.	.	.	.	.	.	.	.	o	.	.	.	o	o	.
Kelly *et al.* (2011)	.	o	.	o	.	.	.	.	.	o	.	o	.	.	.	.	.	o	o	.
Risi *et al.* (2010)	o	.	.	o	.	.	.	.	.	.	.	o	.	.	.	.	.	o	.	.
Asadi *et al.* (2010a)	o	.	.	.	o	.	.	.	.	.	.	.	.	.	o	.	.	o	.	.
Medini *et al.* (2012)	o	.	.	.	.	.	.	.	.	.	o	o	.	.	o	.	.	o	o	.
Binkley *et al.* (2014)	.	o	.	.	.	.	.	.	.	.	o	o	.	.	.	.	.	o	.	.
Yang *et al.* (2014)	.	o	.	.	.	.	.	.	.	.	o	.	.	.	.	o	.	o	.	.
Somasundaram and Murphy (2012)	.	o	.	o	.	.	.	.	.	.	o	.	.	.	.	o	.	.	.	.
Percentage 'o'	53	48	11	14	24	9	11	25	7	7	2	71	1	20	15	17	1	74	19	4

Table B2 Article characterization results of facets 5 and 6. The attributes are described in Table 2. A 'Y' means the article stated that it included this attribute or performed this step; a 'N' means the article stated that it did not include this attribute or perform this task; a '?' means the article did not state either way; and a '-' means this attribute is not applicable to this article.

	Preprocessing								Tools			
	identifiers	comments	strings	granularity	tokenize	stem	stop	prune	tool	K value	justif. of K	iterations
Ahsan *et al.* (2009)	?	?	?	bug report	?	N	Y	Y	MATLAB	50-500	vary	?
Alhindawi *et al.* (2013a)	Y	Y	?	method	Y	?	Y	?	?	?	?	?
Ali *et al.* (2012)	Y	?	?	class/report	?	Y	Y	?	?	50-200	vary	?
Alipour *et al.* (2013)	?	?	?	report	?	?	Y	?	?	35	previous	?
Andrzejewski *et al.* (2007)	?	?	?	?	?	?	?	?	own	?	expert	2000
Antoniol *et al.* (2008)	?	?	?	method	Y	Y	Y	?	own	?	?	?
Asuncion *et al.* (2010)	?	?	?	?	?	Y	Y	?	own	?	?	?
Bajracharya and Lopes (2009)	?	?	?	query	Y	?	N	?	Dragon	50-500	vary	?
Bajracharya and Lopes (2010)	?	?	?	query	Y	?	N	?	Dragon	50-500	vary	?
Baldi *et al.* (2008)	Y	N	N	class	Y	?	Y	?	?	125	vary	?
Bassett and Kraft (2013)	Y	Y	Y	?	Y	Y	Y	Y	?	?	?	?
Bavota *et al.* (2010)	?	?	?	method	?	?	?	?	?	?	?	?
Bavota *et al.* (2013)	Y	?	?	class/req.	Y	Y	Y	?	?	?	vary	?
Beard *et al.* (2011)	Y	Y	?	method	?	Y	Y	?	Gensim	75	vary	200
Binkley *et al.* (2012)	Y	?	?	class	Y	?	Y	?	?	?	?	?
Bose and Suresh (2008)	?	?	?	?	?	?	?	?	?	?	?	?
Campbell *et al.* (2013)	?	?	?	req./post	Y	?	Y	?	TMT	400	?	?
Capobianco *et al.* (2009)	?	?	?	class	?	?	Y	?	?	?	?	?
Chen *et al.* (2012)	Y	Y	Y	class	Y	Y	Y	Y	MALLET	500	previous	10000
Cleary *et al.* (2008)	Y	Y	Y	?	Y	Y	Y	Y	?	300	?	?
Corley *et al.* (2012)	Y	Y	?	class	Y	?	Y	?	MALLET	50-200	?	1000
Dasgupta *et al.* (2013)	Y	Y	?	class/report	Y	?	?	?	TraceLab	?	?	?
De Lucia *et al.* (2011)	Y	Y	?	class/req.	Y	Y	Y	?	?	?	?	?
de Boer and van Vliet (2008)	?	?	?	req.	N	?	Y	?	?	5	?	?
De Lucia *et al.* (2004)	Y	?	?	?	Y	?	Y	?	own	?	?	?
De Lucia *et al.* (2006)	?	?	?	?	?	N	?	?	own	?	?	?
De Lucia *et al.* (2007)	Y	N	N	?	Y	?	Y	Y	own	?	?	?
Dit *et al.* (2008)	?	?	?	bug report	Y	?	Y	?	?	300	?	?
Dit *et al.* (2013a)	Y	?	?	class/req.	Y	Y	Y	?	TraceLab	?	vary	?
Eddy *et al.* (2013)	?	?	?	class	Y	?	Y	?	?	?	?	?
Eyal-Salman *et al.* (2013)	Y	Y	?	class/req.	Y	Y	?	?	?	?	?	?
Gall *et al.* (2008)	Y	Y	?	class	?	?	?	?	own	?	?	?
Gethers and Poshyvanyk (2010)	Y	Y	?	class	Y	?	?	?	lda-r	75, 125, 225	vary	?
Gethers *et al.* (2011b)	Y	Y	?	class	?	?	Y	?	lda-r	?	?	?
Gethers *et al.* (2011d)	Y	?	?	class	Y	Y	Y	Y	lda-r	?	vary	?
Gethers *et al.* (2012)	Y	Y	?	method/report	Y	Y	?	Y	?	?	?	?
Gethers *et al.* (2011a)	Y	Y	?	method/report	?	?	?	?	?	?	?	?
Grant *et al.* (2008)	Y	Y	Y	method	?	?	?	Y	own	10	?	?
Grant and Cordy (2009)	Y	N	Y	method	?	?	?	Y	?	?	?	?
Grant and Cordy (2010)	Y	N	?	method	Y	?	?	?	GibbsLDA	50-300	vary	?

continued on next page

continued from previous page

	Preprocessing								Tools			
	identifiers	comments	strings	granularity	tokenize	stem	stop	prune	tool	K value	justif. of K	iterations
Grant *et al.* (2011b)	Y	Y	Y	class/method	?	?	?	?	Multiple	100	?	?
Grant *et al.* (2011a)	?	?	Y	WSDL	Y	?	?	?	?	100	?	?
Han *et al.* (2012)	?	?	?	report	Y	?	Y	?	TMT	10-70	manual	?
Hayes *et al.* (2006)	?	?	?	?	?	Y	Y	?	own	10-100	vary	?
Hindle *et al.* (2009)	?	?	?	commit msg	?	?	Y	Y	lda-c	20	vary	?
Hindle *et al.* (2010)	?	?	?	commit msg	?	?	?	?	?	?	?	?
Hindle *et al.* (2011)	?	?	?	log	?	?	?	Y	?	20	previous	?
Hindle *et al.* (2012c)	?	?	?	req.	Y	Y	Y	?	?	5-250	vary	?
Hu and Wong (2013)	?	Y	?	class/log	?	?	?	?	?	20	previous	?
Iacob and Harrison (2013)	?	?	?	review	?	?	?	?	?	5	?	?
Islam *et al.* (2012b)	?	?	?	class/req.	?	?	?	?	?	300	?	?
Jiang *et al.* (2008)	?	?	?	?	?	?	?	?	own	?	?	?
Kagdi *et al.* (2010)	Y	Y	?	method	?	?	?	?	?	?	?	?
Kagdi *et al.* (2012a)	Y	Y	?	class	Y	Y	Y	?	?	?	?	?
Kaushik *et al.* (2011)	Y	Y	?	class	Y	Y	Y	?	?	50-500	vary	?
Kaushik and Tahvildari (2012)	?	?	Y	report	Y	Y	Y	?	Gensim	400-550	vary	?
Kawaguchi *et al.* (2006)	Y	N	N	system	?	?	?	Y	own	?	?	?
Kouters *et al.* (2012)	?	?	?	log	?	?	?	?	?	?	?	?
Kuhn *et al.* (2005)	Y	Y	?	class/method	Y	Y	Y	?	?	200-500	?	?
Kuhn *et al.* (2007)	Y	Y	?	class	Y	Y	Y	?	own	15	?	?
Kuhn *et al.* (2008)	?	?	?	class	?	?	?	?	own	?	?	?
Kuhn *et al.* (2010)	Y	Y	Y	class	?	?	?	?	own	50	?	?
Lin *et al.* (2006)	N	Y	N	class	?	Y	Y	?	own	?	?	?
Linstead *et al.* (2007b)	?	?	?	class	Y	?	Y	?	own	100	vary	3000
Linstead *et al.* (2007a)	?	?	?	?	Y	?	Y	?	TMT	100	vary	3000
Linstead *et al.* (2008b)	?	?	?	?	Y	?	Y	?	?	100	vary	3000
Linstead *et al.* (2008c)	?	?	?	?	Y	?	Y	?	?	100	vary	3000
Linstead *et al.* (2008a)	Y	?	?	class	Y	?	Y	?	?	100	vary	?
Linstead and Baldi (2009)	?	?	?	bug report	Y	?	Y	?	?	100	vary	3500
Linstead *et al.* (2009)	?	?	?	?	?	?	?	?	?	?	?	?
Liu *et al.* (2009)	Y	Y	?	method	Y	?	Y	?	GibbsLDA	100	?	1000
Lohar *et al.* (2013)	?	?	?	class/report/req.	Y	Y	Y	?	?	?	?	?
Lormans and Van Deursen (2006)	?	?	?	?	?	Y	Y	?	TMG	?	?	?
Lormans *et al.* (2006)	?	?	?	?	?	?	?	?	own	?	?	?
Lormans (2007)	?	?	?	?	?	?	?	?	own	?	?	?
De Lucia *et al.* (2012)	Y	Y	?	class	Y	Y	Y	Y	?	?	vary	?
De Lucia *et al.* (2014)	Y	Y	?	class	Y	Y	Y	?	?	?	previous	?
Lukins *et al.* (2008)	Y	Y	Y	method	Y	Y	Y	?	GibbsLDA	100	?	?
Lukins *et al.* (2010)	Y	Y	?	method	N	N	N	?	GibbsLDA	100	vary	?
Maletic and Valluri (1999)	Y	Y	Y	class/method	?	?	?	?	?	250	vary	?

continued on next page

continued from previous page

	Preprocessing								Tools			
	identifiers	comments	strings	granularity	tokenize	stem	stop	prune	tool	K value	justif. of K	iterations
Maletic and Marcus (2001)	Y	Y	Y	class/method	?	?	?	?	?	350	?	?
Marcus and Maletic (2001)	Y	Y	Y	class/method	?	?	?	?	own	350	?	?
Marcus and Maletic (2003)	Y	Y	Y	class	Y	?	?	?	?	?	?	?
Marcus *et al.* (2004)	Y	Y	N	?	Y	?	?	?	?	?	?	?
Marcus (2004)	Y	Y	?	class/method	Y	?	?	?	?	?	?	?
Marcus *et al.* (2005)	Y	Y	N	?	?	?	?	?	?	?	?	?
Marcus *et al.* (2008)	Y	Y	N	method	?	?	?	?	?	?	?	?
Maskeri *et al.* (2008)	Y	Y	?	class	Y	Y	Y	?	own	30	?	?
McMillan *et al.* (2009)	Y	?	?	method	Y	Y	Y	?	own	25-75	?	?
Misra *et al.* (2012)	Y	Y	?	class	Y	Y	Y	?	?	30	ratio	?
Moritz *et al.* (2013)	?	?	?	method	?	?	?	?	?	?	?	?
Naguib *et al.* (2013)	?	?	?	report	Y	?	Y	?	MALLET	?	?	?
Neuhaus and Zimmermann (2010)	?	?	?	report	N	Y	Y	?	?	40	?	?
Nguyen *et al.* (2011b)	Y	Y	Y	class	Y	?	Y	?	?	5	?	50
Nguyen *et al.* (2011a)	Y	Y	?	class/report	Y	Y	Y	Y	own	1-1000	vary	?
Nguyen *et al.* (2012)	?	?	?	report	?	Y	Y	?	?	20-400	vary	?
Nie and Zhang (2012)	Y	Y	?	class	?	Y	Y	?	?	?	vary	?
Niu *et al.* (2012)	Y	Y	?	class	?	?	?	?	?	?	?	?
Oliveto *et al.* (2010)	?	?	?	?	?	Y	Y	Y	?	250	vary	?
Oliveto *et al.* (2011)	?	?	?	class/method	?	?	?	?	?	?	?	?
Ossher *et al.* (2009)	?	?	?	?	?	?	?	?	?	?	?	?
Panichella *et al.* (2013)	?	?	?	class	?	?	?	?	?	10-500	vary	500
Poshyvanyk *et al.* (2006)	Y	Y	?	method	?	?	?	?	?	?	?	?
Poshyvanyk and Marcus (2007)	Y	Y	N	method	Y	N	N	N	?	?	?	?
Poshyvanyk *et al.* (2007)	Y	Y	N	method	Y	N	N	N	?	500	?	?
Poshyvanyk and Grechanik (2009)	?	?	?	?	?	?	?	?	?	?	?	?
Poshyvanyk *et al.* (2013)	Y	Y	?	class/method	Y	Y	Y	?	?	?	?	?
Qusef *et al.* (2013)	Y	Y	?	class	?	?	?	?	lda-r	?	?	?
Revelle and Poshyvanyk (2009)	Y	?	?	method	?	?	?	?	?	?	?	?
Revelle *et al.* (2010)	Y	Y	Y	method	Y	Y	?	?	?	?	?	?
Saha *et al.* (2013)	Y	Y	Y	class/report	Y	Y	Y	?	Indri	?	?	?
Savage *et al.* (2010)	Y	Y	?	class	Y	Y	Y	?	JGibbLDA	input	?	input
Shang *et al.* (2013)	?	?	?	log	?	?	?	?	MALLET	5	?	?
Sharafl *et al.* (2012)	Y	Y	?	class	Y	Y	Y	?	MALLET	2	?	?
Thomas *et al.* (2010)	Y	Y	?	class	Y	Y	Y	?	MALLET	45	previous	?
Thomas *et al.* (2013)	Y	Y	?	class/report	Y	Y	Y	?	MALLET	32-256	vary	max
Tian *et al.* (2009)	Y	Y	?	system	Y	?	Y	?	GibbsLDA	40	vary	?
Ujhazi *et al.* (2010)	Y	Y	N	method	Y	Y	Y	?	?	?	?	?
Van der Spek *et al.* (2008)	Y	Y	N	method	Y	N	Y	?	SVDLIBC	input	vary	?
Wang *et al.* (2011)	Y	Y	?	method	Y	Y	Y	?	?	50-500	vary	?

continued on next page

continued from previous page

	Preprocessing								Tools			
	identifiers	comments	strings	granularity	tokenize	stem	stop	prune	tool	K value	justif. of K	iterations
Wu *et al.* (2008)	?	?	?	log	Y	Y	Y	?	JAMA	?	?	?
Xia *et al.* (2013)	?	?	?	report	?	Y	Y	?	JGibbLDA	5% of unique terms	ratio	500
Xie *et al.* (2013)	Y	?	?	class	Y	Y	Y	?	JGibbLDA	?	max likelihood	2000
Xue *et al.* (2012)	Y	Y	Y	class	Y	Y	Y	?	?	?	ratio	?
Zawawy *et al.* (2010)	?	?	?	log	?	Y	Y	?	?	?	?	?
Zhou *et al.* (2012)	Y	?	?	class/report	Y	Y	Y	?	JGibbLDA	100-500	?	?
Medini (2011)	?	?	?	log	?	?	?	?	?	?	?	?
Zou and Hou (2014)	?	?	?	?	?	?	?	?	?	?	?	?
Limsettho *et al.* (2014)	?	?	?	report	Y	Y	Y	?	?	25-600	?	?
Grant and Cordy (2014)	?	?	?	?	?	?	?	?	GibbsLDA	?	ratio	?
Yu (2012)	?	?	?	report	Y	Y	Y	?	?	200	previous	100000
Thomas *et al.* (2014)	Y	Y	Y	class	Y	Y	Y	?	MALLET	21-48	ratio	200
Grant *et al.* (2012)	?	?	?	method	?	?	?	?	?	25-650	vary	?
Bavota *et al.* (2014)	Y	Y	Y	method	Y	Y	Y	?	lda-r	75	previous	?
Alhindawi *et al.* (2013b)	?	?	?	class/req.	Y	Y	Y	Y	?	?	?	?
Parizy *et al.* (2014)	Y	Y	?	method	Y	Y	Y	Y	?	50	?	?
Islam *et al.* (2012a)	Y	Y	?	class/req.	Y	?	Y	?	?	100	?	?
Le *et al.* (2013)	Y	Y	Y	method/report	Y	Y	?	?	?	?	?	?
Misra and Das (2013)	?	?	?	req.	?	Y	Y	Y	?	?	?	?
Asadi *et al.* (2010b)	Y	Y	?	method/log	Y	Y	Y	Y	?	50	?	?
Dit *et al.* (2013b)	Y	?	?	method/req.	Y	Y	?	?	?	300	previous	?
Tairas and Gray (2009)	Y	N	?	class	?	N	N	?	MATLAB	?	previous	?
Lormans *et al.* (2008)	?	?	?	test/req.	?	Y	?	?	?	40%	?	?
Ali *et al.* (2014)	Y	Y	?	class/req.	Y	Y	Y	Y	MALLET	2-100	vary	?
Kagdi *et al.* (2012b)	Y	Y	?	method	?	?	Y	?	?	300	?	?
Bavota *et al.* (2012)	?	?	?	method	?	?	?	?	?	?	?	?
Raja (2012)	?	?	?	report	?	Y	Y	?	?	?	?	?
Barua *et al.* (2012)	?	?	?	discussion	?	Y	Y	Y	MALLET	40	vary	500
Hindle *et al.* (2014)	?	Y	?	req.	Y	Y	Y	?	?	10-250	vary	1000
Hindle *et al.* (2012a)	?	?	?	log	?	N	Y	?	?	20	?	?
Pagano and Maalej (2013)	?	?	?	blogs	?	Y	Y	?	?	50	vary	?
Biggers *et al.* (2014)	Y	Y	Y	method	Y	Y	Y	?	MALLET	75-200	vary	?
Canfora *et al.* (2014)	?	?	?	log	?	Y	Y	?	lda-r	10	vary	?
Gorla *et al.* (2014)	?	?	?	app description	?	Y	Y	Y	?	30	ratio	?
Linares-Vásquez *et al.* (2013)	?	?	?	discussion	Y	Y	Y	?	FastLDA	20	?	1000
Allamanis and Sutton (2013)	Y	?	?	discussion	Y	Y	?	?	MALLET	150	?	2000
Bajaj *et al.* (2014)	?	?	?	discussion	?	Y	Y	?	?	?	?	?
Brickey *et al.* (2012)	?	?	?	survey	?	?	?	?	?	?	?	?
Pingclasai *et al.* (2013)	?	?	?	report	Y	Y	Y	?	?	10-150	vary	?
Galvis Carreño and Winbladh (2013)	?	?	?	req./comment	Y	?	Y	?	?	25-150	manual	?
Kelly *et al.* (2011)	Y	N	?	class	Y	?	?	?	MALLET	20	?	?
Risi *et al.* (2010)	Y	Y	?	class	Y	Y	Y	Y	?	?	?	?
Asadi *et al.* (2010a)	Y	Y	?	method/log	Y	Y	Y	Y	?	50	?	?
Medini *et al.* (2012)	?	Y	?	method/log	Y	Y	Y	Y	?	50	previous	?
Binkley *et al.* (2014)	Y	Y	?	class	Y	?	?	?	?	5-300	vary	?
Yang *et al.* (2014)	?	?	?	report	Y	Y	Y	?	TMT	30	?	?
Somasundaram and Murphy (2012)	?	?	?	report	Y	?	Y	?	?	20-270	?	?
Percentage 'Y'	53	44	13	-	56	46	60	15	-	-	-	-
Percentage 'N'	1	4	7	-	2	5	4	1	-	-	-	-
Percentage '?'	46	52	80	14	43	50	36	84	58	45	62	87