Changeset-Based Topic Modeling of Software Repositories

Christopher S. Corley, Kostadin Damevski, Nicholas A. Kraft

Abstract—The standard approach to applying text retrieval models to code repositories is to train models on documents representing program elements. However, code changes lead to model obsolescence and to the need to retrain the model from the latest snapshot. To address this, we previously introduced an approach that trains a model on documents representing changesets from a repository and demonstrated its feasibility for feature location. In this paper, we expand our work by investigating: a second task (developer identification), the effects of including different changeset parts in the model, the repository characteristics that affect the accuracy of our approach, and the effects of the time invariance assumption on evaluation results. Our results demonstrate that our approach is as accurate as the standard approach for projects with most changes localized to a subset of the code, but less accurate when changes are highly distributed throughout the code. Moreover, our results demonstrate that context and messages are key to the accuracy of changeset-based models and that the time invariance assumption has a statistically significant effect on evaluation results, providing overly-optimistic results. Our findings indicate that our approach is a suitable alternative to the standard approach, providing comparable accuracy while eliminating retraining costs.

Index Terms—changesets; feature location; developer identification; program comprehension; mining software repositories; online topic modeling

1 Introduction

Researchers have identified numerous applications for text retrieval (TR) models in facilitating software maintenance tasks. TR-based techniques are particularly appropriate for problems which require retrieval of software artifacts from large software repositories, such as feature location [1], code clone detection [2] and traceability link recovery [3]. Due to the rapid pace of change and the large scale of modern software repositories, TR-based techniques must be compatible with continual software change if they are to retrieve accurate and timely results.

The standard methodology for applying a TR model to a source code repository is to extract a document for each file, class, or method in a source code snapshot (e.g., a particular release of a project), to train a TR model on those documents, and to create an index of the documents from the trained model [1]. Topic models are a class of TR models that includes latent Dirichlet allocation (LDA) and has been applied widely within software engineering, to problems such as feature location [4], [5] and command prediction in the IDE [6].

Unfortunately, topic models like LDA cannot be updated to accommodate the addition of new documents or the modification of existing documents, and thus these topic models must be repeatedly retrained as the input corpus evolves. Online topic models, such as online LDA [7], natively support the online addition of new documents, but they still cannot accommodate modifications to existing documents. Consequently, applying a TR model to a rapidly evolving source code repository using the standard methodology incurs substantial (re)training costs that are incompatible with the goal of integrating TR-based techniques into the IDE.

To address the shortcoming of the standard methodology, we introduced a new methodology based on changesets [8]. Our methodology is to extract a document for each changeset in the source code history, to train a TR model on those changeset documents, and to create an index of the files, classes, or methods in a system from the trained (changeset) model. The methodology stems from the observations that a changeset contains program text (like a file or class/method definition) and is immutable (unlike a file or class/method definition). That is, the changesets for a project represent a stream of immutable documents that contain program text, and thus can be used as input for an online topic model.

Using changesets as the basis of a text retrieval model in software engineering is a novel idea, which, in addition to better adapting to software change, could present certain additional qualitative advantages. While the typical program-element-as-document representations encode program structure into the model, changesets encode frequently changed code, prioritizing the change-prone areas of the source code that may be more relevant for software maintenance tasks than the less frequently changed areas. Note also that there is no loss in fidelity in using changesets, only a difference in what the model prioritizes, as the complete set of changesets in a software repository contains a superset of the program text that is found in a single repository

- *C.S. Corley is with the Department of Computer Science, The University of Alabama, Tuscaloosa, AL, 35487, U.S.A. E-mail: cscorley@crimson.ua.edu*
- *K. Damevski is with the Department of Computer Science, Virginia Commonwealth University, Richmond, VA, 23284, U.S.A. E-mail: damevski@acm.org*
- *N.A. Kraft is with ABB Corporate Research, Raleigh, NC, 27606, U.S.A. E-mail: nicholas.a.kraft@us.abb.com*

Manuscript received February 23, 2018.

snapshot (that is used to build typical text retrieval models for source code).

In our previous work [8], we evaluated our new methodology by comparing two feature location techniques (FLTs) — one based on the standard methodology and one based on our new methodology — using 600 defects and features from four open-source Java projects. In this paper we expand the investigation of our changeset-based methodology by examining the:

- Applicability of our methodology to two software maintenance tasks: feature location and developer identification
- Configurations of changesets that are most appropriate for use with our methodology
- Characteristics of software repositories which cause our methodology to produce better/worse results

We conduct our study using over 750 defects and features from six open-source Java projects for feature location and over 1,000 defects and features from those same Java projects for developer identification.

We also examine the effect of the typical time inconsistency present in the evaluation of text retrieval models for software maintenance. Time inconsistency stems from the fact that, in prior evaluations, researchers evaluate queries using a snapshot of the software that comes much later than the one(s) for which the maintenance tasks were active. For instance, evaluations commonly consider all defects or feature requests submitted between release A and B, with the queries being issued against release B. This assumption allows the use of a single TR model, trained on release B, rather than the use of many TR models trained on the versions associated with each defect or feature request submission. However, this assumption implicitly asserts that changes to the software that occur between the submission of a particular defect or feature request and the release on which the evaluation is performed do not affect the evaluation results. We observe that there are in fact statistically significant differences in evaluation results absent the *time invariance assumption* in prior research and that evaluations that use a time invariance assumption can overstate the performance of the evaluated technique.

The rest of this paper is organized as follows. We begin by describing the background and related work for topic modeling and the two software maintenance applications of interest, feature location and developer identification, in Section 2. Next, in Section 3, we describe how we build topic models from changesets. Section 4 lists the results of applying this model, including an analysis of various configuration options and a discussion of project characteristics that work in favor (or against) our methodology. Section 5 discusses the threats to validity of our experiments. Finally, Section 6 presents the conclusions and future work of this paper.

2 Background & Related Work

Changesets represent developer commits into a repository, consisting of the lines (of source code or other text) removed and added, lines of context surrounding where the change is to be applied, and a natural language commit message describing the change. While changesets do not typically contain the full text of the program, they contain a complete representation of a specific change in the code. As an example, consider the changeset in Figure 1, which addressed Issue #1689 in the Tika open source software.

Topic modeling is a family of dimensionality reduction techniques that extract a set of topics from a corpus of documents. The topics represent co-occurring sets of words, which ideally correspond to an abstract human-recognizable concept. For instance, one extracted topic from a corpus of news articles may contain words like ”flood”, ”wind”, ”storm”, ”rain” related to the concept of weather. Latent Dirichlet Allocation (LDA) [9] is a specific topic modeling technique that has been found to be useful in a variety of applications.

A typical application of topic modeling is text retrieval, where the extracted topic model is used to match a query to documents in the corpus. Two popular text retrieval applications in software engineering are feature location, where a developer searches for code elements relevant to a maintenance task, and developer identification, where a maintenance task is mapped to the most appropriate developer.

In the following, we first describe LDA and then describe work related to feature location and developer identification.

2.1 Latent Dirichlet Allocation

Latent Dirichlet Allocation (LDA) [9] is a probabilistic model that extracts a interpretable representation from a large high-dimensional dataset. It is commonly applied to documents containing natural language text. In this context, given a set of documents LDA extracts a latent document-topic distribution and a corresponding topic-word distribution. LDA is considered to be a generative model, which is to say that by sampling from the extracted document-topic distribution and topic-word distributions one should be able to generate the original corpus. Of course, the goal of LDA is not to generate new documents from these distributions, but instead, produce a lower-dimensional interpretable model that is also capable of inferring the topic distributions of previously unobserved documents.

LDA does not consider the order of words in each document, i.e. uses a bag-of-words assumption. It models each document as a probability distribution indicating the likelihood that it expresses each topic and models each topic that it infers as a probability distribution indicating the likelihood of a word from the corpus coming from the topic.

As a Bayesian model, LDA is configured via a small set of priors, which influence how the model is extracted from the input corpus. For instance, the hyperparameters α and β influence the “smoothness” of the model. Hyperparameter α influences the topic distribution per document, and hyperparameter β influences the word distribution per topic. For example, lowering β results in each topic becoming more specific (i.e., a topic is likely to consist of words not in any other topics), while increasing β causes each topic to become more general (i.e., it causes words to begin to appear across multiple topics). Likewise, lowering α causes each document to express less topics while raising α causes documents to relate to more topics.

```
commit b1432f097ada17573c2dbf81e982915e3e81c815
Author: Tim Allison <########@apache.org>
Date:   Fri Jul 24 18:22:47 2015 +0000

    TIKA-1689: revert mistakenly flipped sort order of parsers from r1677328

    git-svn-id: https://svn.apache.org/repos/asf/tika/trunk@1692564 13f79535-47bb-0310-9956-ffa450edef68

diff --git a/tika-core/src/main/java/org/apache/tika/utils/ServiceLoaderUtils.java b/tika-core/src/main/java/org/apache/
    tika/utils/ServiceLoaderUtils.java
index ef278808..5ee1fe86 100644
--- a/tika-core/src/main/java/org/apache/tika/utils/ServiceLoaderUtils.java
+++ b/tika-core/src/main/java/org/apache/tika/utils/ServiceLoaderUtils.java
@@ -38,9 +38,9 @@ public class ServiceLoaderUtils {
                 if (t1 == t2) {
                     return n1.compareTo(n2);
                 } else if (t1) {
-                    return 1;
-                } else {
                     return -1;
+                } else {
+                    return 1;
                 }
             }
         });
```

Fig. 1. Example of a `git diff`. The first half (in orange), shows the commit id, author name, date of commit, and the message associated with the change, followed by the actual diff of the change. Green lines (beginning with a single +) denote added lines, and red lines (beginning with a single −) denote removed lines. Black or blue lines denote metadata about the change useful for applying the patch. In particular, black lines (beginning with a single space) represent context lines.

To use LDA in a search application, we transform the query q into a topic probability distribution. First, we vectorize q into a vector of term frequencies. Next, we infer from the model the topic probability distribution for the query. We use the resulting distribution to make pairwise comparisons against all documents in the corpus.

Hoffman et al. [7] introduced a variant of LDA which uses online learning from a stream of documents, instead of from a static corpus. This is the variant we leverage in our approach to modeling software repositories.

2.2 Feature Location

Feature location is the act of identifying the source code entity or entities that implement a feature [10]. It is a frequent and fundamental activity for a developer tasked with maintaining a software system. Whether a maintenance task involves adding, modifying, removing a feature, or fixing a bug, a developer cannot complete the task without first locating the source code that implements the feature [11], and both technical debt [12] and code clones [13] can increase the difficulty of manual feature location.

Marcus et al. [1] used a feature location technique (FLT) based on the Latent Semantic Indexing (LSI) [14] topic model to find concepts in the code based on queries from the user, finding that concepts were identifiable from developer-specified identifiers. LSI-based FLTs were subsequently proposed by several others [15], [16], [17], [18], [19].

Lukins et al. [4] introduce an FLT based on latent Dirichlet allocation (LDA) [9] and found that it outperforms the LSI-based FLT by Poshyvanyk et al. [20]. They used LDA's inference technique to infer the topic distributions of queries, i.e., bug reports. Later, they showed LDA to be appropriate for software systems of any size [21].

Biggers et al. [22] investigated the configuration parameters for an LDA-based FLT. They show that excluding source code text such as comments and literals negatively impacts the accuracy of the FLT. Most importantly, they showed that configuration parameters taken from the machine learning and natural language processing (NLP) communities are not good choices for software. Dit et al. [23] showed the need for better term splitting techniques for software text retrieval. Corley et al. [24] examined using deep learning techniques for feature location.

Bassett and Kraft [25] presented new term weighting schemes based on the structural information available in source code. Namely, they found that increasing the weight of method names increases the accuracy of an LDA-based FLT. A typical weighting scheme from the NLP communities is term frequency-inverse document frequency (tf-idf) [26]. Saha et al. [27] show that using structural information provides improvement over tf-idf, as well.

Rao et al. [28] also target the problem of building topic models, introducing an incremental framework for bug localization. Bug localization is the process of identifying source code entities that implement a bug, or an unwanted feature [21]. Although practical, the approach involves using an extended topic modeler to allow updating, adding, and removing documents from the model and index post-hoc. While the approach is essentially equivalent to topic modeling in batch, Rao et al. [28] notes that these algorithm modifications have limitations and thus models may need to be periodically retrained.

2.3 Developer Identification

Developer identification is a triaging activity in which a team member identifies a list of developers that are most apt to complete a change request and assigning one or more of those developers to the task [29]. Begel et al. [30] show that developers need help finding expertise within their organization more than they need help finding source code elements.

As noted by Shokripour et al. [31], there are two broad categories of work in this area: activity-based approaches and location-based approaches. Activity-based developer

identification techniques (DIT) use information gained from a developers activity, e.g., which change requests they have worked on in the past, while location-based DITs rely on source code entity information to derive a developer, e.g., which developer has worked on the related classes in the past. The location-based techniques present an opportune application area for our source code modeling approach.

There are multiple ways to determine the ownership of source code entities. McDonald and Ackerman [32] present a heuristic-based recommender system named Expertise Recommender that uses heuristics derived in a previous industrial study on how developers locate expertise [29]. The Expertise Recommender considers developers' expertise profile based on who last changed a module, who is closest to the requester in the organization, and how connected the requester and expert are by using social network analysis.

Minto and Murphy [33] propose a DIT that uses multiple matrices representing file dependency, or how often pairs of files change together, and file authorship, or how often a developer changes a file. They evaluate the tool on the history of three open source projects: Bugzilla, Eclipse, and Firefox and obtain higher precision and recall compared to the approach proposed by McDonald and Ackerman [32].

Kagdi et al. [34] present a tool named xFinder to mine developer contributions in order to recommend a ranked list of developers for a change. The tool measures the similarity of vectors consisting of the number of commits to a file, the number of workdays spent on a file, and the most recent workday on the file. To find an appropriate developer for a file, they measure similarity between each developer's vector and the file vector. Bird et al. [35] find that measuring ownership in this way correlates low ownership with post-release defects.

Linares-Vasquez et al. [36] present an approach that does not require mining the software history. Using the author indicated in source code comments with a topic model-based FLT, they are able to identify the correct developer. Hossen et al. [37] extend this approach to also include change proneness to adjust the rank of relevant source code entities before selecting a developer. Tamrawi et al. [38] present an incremental DIT approach based on fuzzy sets.

3 Building Topic Models from Changesets

In this section, we will contrast the typical repository snapshot-based topic model construction to that proposed in this paper that relies on a stream of changesets. Text retrieval pipelines consist of a set of indexing steps (left side of Figure 2), performed once, and any number of subsequent retrieval steps where the index is queried with a user specified query (right side of Figure 2).

As a first step in indexing a software repository snapshot, the Document Extractor processes the raw data (i.e., files primarily consisting of source code) and produces a corpus as output, by normalizing words to lowercase, stemming [39], or other similar pre-processing steps. Subsequently, the Topic Modeler executes the topic model learning algorithm (e.g., Gibbs sampling) to extract the topic model. The Indexer uses the topic model to infer the topic distribution for each document from the corpus, which it stores in the Index for fast lookup.

The right side of Figure 2 illustrates the retrieval process. The main component of the retrieval process is the Search Engine, which matches a user-specified query in the Index. Search engines based on topic models also need the trained model in order to infer the topic distribution of the query prior to matching it. The primary function of the search engine is to produce a ranked list of documents based on their (topic) similarity to the query, based on one of many similarity measures. For comparing the topic distributions of the query and documents in the index, we typically rely on metrics appropriate for contrasting discrete probability distributions, e.g. Hellinger distance.

The overall difference in our approach and the standard (snapshot-based) approach to build text retrieval systems based on source code is minimal, shown by the area highlighted in Figure 2. In order to use changesets to train the topic model we only need to make a single adaptation to the pipeline. That is, at the Topic Modeler stage, instead of using only a snapshot of the repository, we use changesets as documents and train the model using an online variant of LDA. The remainder of the pipeline is the same, and the Indexer uses the Snapshot Corpus in order to index program elements at the granularity they are to be retrieved.

The key intuition is that a topic model such as LDA can infer any document's topic proportions regardless of the documents used to train the model, as long as they share the same vocabulary of words. Note that we do not construct an index of the changeset documents used to train the model. We only use the changesets to continuously update the topic model and only use the repository snapshot for indexing. To leverage the online functionality of the topic models, we can also intermix the model training, indexing, and retrieval steps. We can also perform these steps incrementally as a project evolves. This is practical because inferencing a document's topics is a fairly lightweight operation with most topic models. On the other hand, the initial learning performed by the Topic Modeler can be very computationally demanding.

3.1 Rationale for Modeling Repositories Using Changesets

We choose to train the model on changesets, rather than another source of information, because they represent what we are primarily interested in: program features. A single changeset gives us a view of an addition, removal, or modification of a single feature, which may span multiple program elements (i.e. classes and methods). In many cases, a developer can comprehend what a changeset accomplishes by examining it, much like examining a source file.

While a snapshot corpus has documents that represent a program, a changeset corpus has documents that represent programming. If we consider every changeset affecting a particular source code entity, then we gain a sliding-window view of that source code entity over time and the contexts in which those changes took place. Figure 3 shows an example, where light green areas denote added text, dark green denote frequently modified text, while red areas denote text removed in that changeset. Frequently changing program

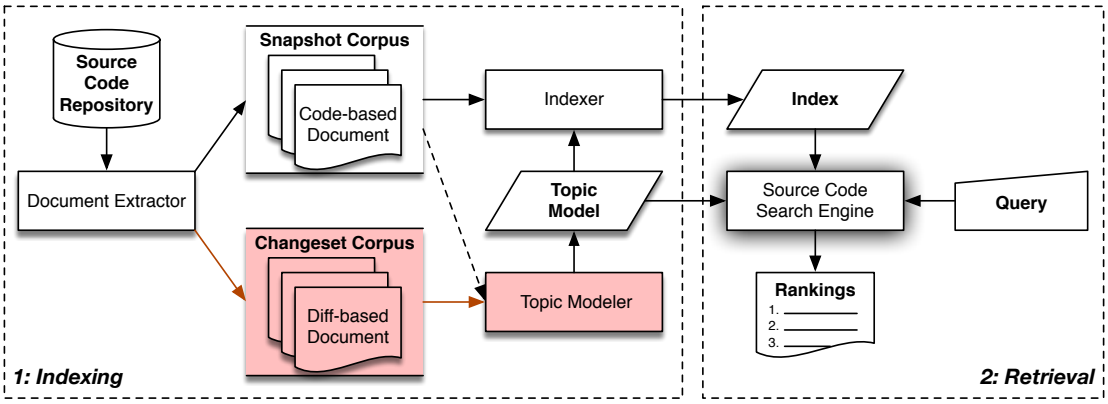


Fig. 2. Differences (highlighted in red) in the architecture of a changeset-based search engine, relative to one based on a snapshot of the source code.

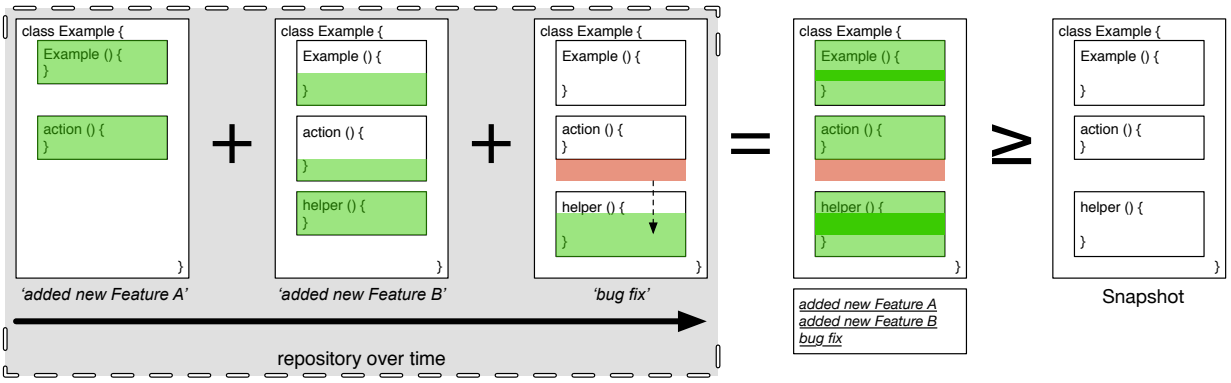


Fig. 3. Changesets over time emphasize frequently changed text while including all of the text in a snapshot.

elements in the repository, which are more likely to be important in future maintenance tasks [40], [41], will be emphasized by a topic model trained on changesets, but not by a model trained on a snapshot corpus. At the same time, the summation of all changes affecting a class over its lifetime would contain all of the words in its current version. In other words, topic modeling a changeset will not miss any information [42].

Using changesets also implies that the topic model may gain some noisy information from these additional documents, especially when considering removals. However, Vasa et al. [40] also observe that code is less likely to be removed than it is to be changed. Also, pure removals can be ignored when building a model based on changesets, which is an option that we experiment with in Section 4. Commit messages add descriptive information to changesets that can provide useful domain context for text retrieval.

Another consideration is that it would appear desirable to remove changesets from the model that are old and no longer relevant. Online LDA already accounts for this by parameterizing the influence newer documents have on the model, thereby decaying the effect of the older documents on the model.

4 Evaluation

In this work, we introduce topic modeling source code repositories in which we incrementally build the model from source code changesets. By using an online variant of LDA trained on changesets, we maintain up-to-date models without incurring the non-trivial computational cost associated with retraining the topic model. In this section we describe the design of our study in which we compare our new methodology with the current practice. For this purpose, we pose the following research questions:

RQ1. Is a changeset-based FLT as accurate as a snapshot-based FLT?

RQ2. Is a changeset-based DIT as accurate as a snapshot-based DIT?

RQ3. Does the time-invariance assumption affect the evaluation of FLT and DIT?

RQ4. What are the effects of using different portions of a changeset for corpus construction, such as additions, removals, context lines, and the commit message?

4.1 Datasets and Benchmarks

For the first two research questions, there do exist various datasets and benchmarks for each [36], [43], [44]. The benchmarks are all extracted from real change tasks in the maintenance history of open source projects. However, these benchmarks all contain a *time invariance assumption*, which may lead to misleading results, resulting from the model constructed from a (much) newer snapshot of the repository than the one that existed during each specific maintenance task. For instance, researchers may extract the benchmarks from maintenance tasks between releases 1.1 and 1.2, while indexing and evaluating only using a repository snapshot

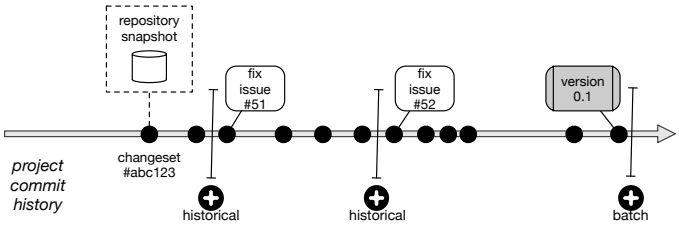


Fig. 4. We study feature location models constructed after a specific release of the software (batch training) and models constructed before a specific issue was closed (historical simulation).

TABLE 1
Subject system corpora and dataset sizes

	Developers	Files	Changesets	Issues
BookKeeper v4.3.0	5	843	574	164
Mahout v0.10.0	38	1556	3283	133
OpenJPA v2.3.0	26	4968	4616	137
Pig v0.14.0	28	2098	2584	222
Tika v1.8	26	954	2469	40
ZooKeeper v3.5.0	16	927	1245	359
Total	139	11346	14771	1055

from release 1.2. In certain cases, the snapshot of the repository is significantly newer than the earliest maintenance task in the gold set. The specific repository snapshot that matches the time when the maintenance task was active is difficult or impossible to recover for these existing datasets. Therefore, for this paper, we create a new dataset that avoids this time invariance assumption that the repository will not substantially change between the time of the maintenance task and the snapshot.

Figure 4 visualizes the commit history of an example software project. Each commit consists of a *changeset* and a *snapshot* of the repository files after the changes are applied. Some of the commits also correspond to the completion of maintenance tasks (i.e. issues in the project tracker), prior to which are moments that the *historical simulation* evaluates the feature location models using the program elements necessary for that specific maintenance task. One of the commits in Figure 4 is tagged with a release of a new version of the software. Following this commit, the batch trained feature location models are evaluated using all of the maintenance tasks in a specific period.

For **RQ1** and **RQ2** we examine changeset trained models using batch training, at a specific release version of a software package, which compares well with existing evaluation approaches but still suffers from the time invariance assumption. For **RQ3** we examine changeset topic models trained as a historical simulation, localized to the state of the repository when the specific maintenance task was performed, with the aim of determining if the time invariance assumption does affect the experimental results.

The six subjects of our studies vary in size and application domain. BookKeeper is a distributed logging service[1]. Mahout is a tool for scalable machine learning[2]. OpenJPA is object-relational mapping tool[3]. Pig is a platform for analyzing large datasets[4]. Tika is a toolkit for extracting metadata and text from various types of files[5]. ZooKeeper is a tool that works as a coordination service to help build distributed applications[6]. Table 1 summarizes the sizes of each system's corpora and dataset.

We chose these systems for our work because developers use descriptive commit messages that allow for easy traceability linking to issue reports. Further, all projects use JIRA as an issue tracker, which has been found to encourage more accurate traceability link recovery [45]. Finally, each system varies in domain and in size, in terms of number of developers, changesets, and source code files.

To build our dataset we mine the Git repository for information about each commit: the committer, message, and files changed. We use the files changed information for the FLT-based evaluation. Using the message, we extract the traceability links to issues in JIRA with the regular expression: `%s-\\d+`, where `%s` is the project's name (e.g., BookKeeper). This matches for JIRA-based issue identifiers, such as BOOKKEEPER-439 or TIKA-42.

From the issue reports, we extract the version the issue marked as fixed in. We ignore issues that are not marked with a fixed version. We also extract the title and description of the issue. For the historical simulation, we also exclude the few issues (e.g. 7 for FLT) where no existing project files were changed, i.e. developers resolved the issue only by adding new files.

We construct two goldsets for each commit linked to an issue report. The first goldset is for evaluating FLTs, and contains the files, classes, and methods changed in the linked commit. The second goldset is for evaluating DITs, and contains the developer(s) that committed those changes[7]. We do not consider whether the change was submitted by a developer and committed by a core contributor. In this case, we assume that the core contributor, as the subject matter expert, understands and agrees with the change.

The number of issues for each project, shown in the fifth column of Table 1, corresponds to the number of searches (i.e., applications of the DIT or FLT) in our study. Over the entire history for each of the six projects, the number of new issues reported each week averages 2.5 for BookKeeper, 3.7 for Mahout, 4.3 for OpenJPA, 9.3 for Pig, 4.6 for Tika, and 5.8 for ZooKeeper. These numbers represent lower bounds for the number of applications of the DIT or FLT in the field. Previous research demonstrates that developers perform feature location as part of their daily activities [11]. Similarly, project members often must perform developer identification more than once for an issue, as issue reports are tossed amongst developers [46], [47].

As a comparison metric, we use Mean Reciprocal Rank (MRR). Reciprocal rank is often useful when there are few documents, or only one, relevant to the query. For our dataset, the median number of files changed per commit is 1 for four of the systems, 1.5 for Tika, and 2 for BookKeeper. Similarly, the median number of developers involved per

1. https://bookkeeper.apache.org
2. https://mahout.apache.org
3. http://openjpa.apache.org
4. http://pig.apache.org
5. http://tika.apache.org
6. http://zookeeper.apache.org

7. Our datasets and scripts are publicly available at: https://github.com/cscorley/triage

change is 1 for all six systems. MRR is an average of reciprocal ranks over different queries, hence it is useful for evaluating the effectiveness of a search engine [48]. We also use the Wilcoxon signed-rank test with Holm correction to determine the statistical significance of the difference between a pair of rankings. An effect size is calculated using Cliff's delta [49], which ranges from -1 (all values in the first group are larger than the second group) to +1 (all values in the second group are larger than the first group). A value of zero indicates that the two groups are identical. We examined the effect size with the criteria of $|\delta| > 0.147$ = small effect, $|\delta| > 0.33$ = medium effect, and $|\delta| > 0.474$ = large effect [50].

4.2 Feature Location (RQ1)

To answer **RQ1**, we evaluate two batch-trained models and a model trained as a historical simulation. For the batch-trained models, one is trained on a snapshot corpus, and the other on a changeset corpus. The process for these corresponds to Figure 2. Both models index a snapshot of the corpus that is newer than the last query (or issue), corresponding to the next tagged version of the project. This may in fact be significantly later than just after the last issue.

RQ1 asks how well a topic model trained on changesets performs compared to one trained on source code entities. Table 2a summarizes the results of each subject system when evaluated at the file-level. In the table, we bold the greater of the two MRRs. Since our goal is to show that training with changesets is just as good, or better than, training on snapshots, we only care about statistical significance when the MRR is in favor of snapshots. While statistical significance in favor of changesets is desirable, statistical insignificance between snapshots and changesets is acceptable and also desirable as it showcases that the changeset approach is on par with snapshots. For example, Pig is a favorable case for changesets as it has a higher MRR, along with statistical significance ($p < 0.01$) and the largest effect size for FLT (0.1285). Likewise, Tika displays a favorable case for snapshots in terms of higher MRR, but does not achieve statistical significance and hence it is not a definite unfavorable case.

We note an improvement in MRR for 4 of the 6 systems when using changesets. Mahout is the only system with an MRR in favor of snapshots and statistically significant at $p < 0.01$. For Mahout, however, the effect size is very small and difference in MRR is negligible (2.54%). Comparing all systems at once by combining all effectiveness measures, changesets show slight MRR improvement over snapshots with statistical significance. This suggests that changeset-based topic models are on par with snapshot-based topic models, and for the majority of systems, the better choice.

4.3 Developer Identification (RQ2)

To answer **RQ2**, we take an approach using the developer's profile rather than a heuristic-based approach, which allows for us to construct a more fair evaluation of batch-trained models by only varying the training corpus, as described by Matter et al. [51]. This evaluation requires a snapshot corpus, a changeset corpus, and a corpus of developer profiles.

For developer identification using snapshots, we first build an LDA topic model using the source code snapshot at a specific release version (i.e., like the FLT evaluation). Then, we infer the index of topic distributions with a corpus of developer profiles. For each query in the dataset, we infer the query's topic distribution and rank each developer profile in the index.

For changesets, the process varies only slightly from a snapshot approach, in that we train the model using online training with a batch of changesets. Similarly as before, we infer an index of topic distributions with the developer profiles, and, for each query in the dataset, we infer the query's topic distribution and rank each developer profile.

For the historical simulation, the approach has a few minor differences to the one described for feature location. With each mini-batch, we index the topic distributions with the developer profiles up to that commit. However, each developer profile of any mini-batch includes all developer activity up to that commit, i.e., it is an aggregation of all previous mini-batches, with the addition of activity since the last mini-batch. The remainder of the approach remains the same.

RQ2 asks how well a topic model trained on changesets performs compared to one trained on source code entities. Table 3a summarizes the results of each subject system. While statistical significance in favor of changesets is desirable, statistical insignificance between snapshots and changesets is acceptable and also desirable as it showcases that the changeset approach is on par with snapshots. For example, OpenJPA is a favorable case for changesets as it has a higher MRR, along with statistical significance ($p < 0.01$) and a non-negligible effect size (0.2135). Likewise, Tika displays a favorable case for snapshots in terms of higher MRR, but does not achieve statistical significance and hence it is not a definite unfavorable case. On the other hand, Pig is a system with an MRR in favor of snapshots that is also statistically significant at $p < 0.01$ and has a medium effect size. Comparing all systems at once, snapshots show slight MRR improvement over changesets with statistical significance.

This result is on par with the result we saw for FLT in the previous section, where the advantage was slightly towards changeset-based models. Overall, we see that the comparable results indicate that changeset-based topic models are a suitable replacement to snapshot-based.

4.4 Historical Simulation (RQ3)

We also ask how well a historical simulation of using a topic model would perform as it were to be used in real-time. This is a much closer evaluation of an FLT to it being used in an actual development environment. To conduct the historical simulation, we first determine which commits relate to each query (or issue) and partition mini-batches out of the changesets. We then proceed by initializing a model for online training with no initial corpus, and update the model with each mini-batch. Then, we build an index of topic distributions for the source files at the commit the partition ends on. We also obtain a topic distribution for each query related to the commit. For each query, we rank each source file in the index with pairwise comparison to

Subject System	Queries	MRR Snapshot	MRR Changesets	Spread	Wilcoxon p-value	Effect size
BookKeeper v4.3.0	143	0.4510	**0.4567**	+0.0056	$p = 0.5008$	0.0246
Mahout v0.10.0	50	**0.2984**	0.2730	−0.0254	$p < 0.01$	−0.0820
OpenJPA v2.3.0	131	0.2724	**0.2989**	+0.0265	$p < 0.01$	0.1259
Pig v0.14.0	174	0.3231	**0.3930**	+0.0699	$p < 0.01$	0.1285
Tika v1.8	36	**0.4778**	0.4033	−0.0744	$p = 0.4491$	−0.0756
ZooKeeper v3.5.0	241	0.4742	**0.4818**	+0.0075	$p < 0.01$	0.0618
All	775	0.3907	**0.4092**	+0.0185	$p < 0.01$	0.0726

(a) Snapshot-based vs. Changeset-based Feature Location: MRR, Wilcoxon p-values, and effect size

Subject System	Queries	MRR C-set Batch	MRR C-set Hist.	Spread	Wilcoxon p-value	Effect size
BookKeeper v4.3.0	138	**0.4527**	0.3019	−0.1508	$p < 0.01$	−0.2090
Mahout v0.10.0	50	0.2730	**0.3542**	+0.0812	$p = 0.6074$	0.0748
OpenJPA v2.3.0	131	**0.2989**	0.1513	−0.1476	$p < 0.01$	−0.2823
Pig v0.14.0	174	**0.3930**	0.2501	−0.1428	$p < 0.01$	−0.2173
Tika v1.8	34	**0.3940**	0.3782	−0.0157	$p = 0.1821$	−0.0545
ZooKeeper v3.5.0	241	**0.4818**	0.2983	−0.1835	$p < 0.01$	−0.2450
All	768	**0.4077**	0.2701	−0.1376	$p < 0.01$	−0.1978

(b) Changeset-based Batch vs. Changeset-based Historical Feature Location: MRR, Wilcoxon p-values, and effect size

TABLE 2
Results on Feature Location.

the query's inferred topic distribution. Since our dataset is extracted from the commit that implemented the change, our partitioning is exclusive of that commit. That is, we update the model with the commit immediately preceding the linked commit and infer the snapshot index from the linked commit. This ensures that our evaluations mirror what a developer would encounter when performing an actual feature location task.

Table 2b summarizes the results for each subject system when historical simulation based on changesets is compared to the batch changeset model, which relies on the time invariance assumption. In each of the tables, we bold the greater of the two MRRs. Our goal is to show that temporal considerations must be given during FLT evaluation.

There is an improvement in favor of historical simulation in MRR for only 1 of the 6 systems. Four of the 5 results in favor of batch changesets were statistically significant. Overall, batch-trained changeset-based topic models perform better than a full historical simulation with statistical significance. This indicates that the two types of evaluations are indeed different. The results of our historical simulation evaluation reveal that the accuracy of the changeset-based FLT is inconsistent as a project evolves, and is actually lower than indicated by batch evaluation.

Table 3b summarizes the results of using historical simulation for DIT. Again, our goal here is only to show that temporal considerations must be given during DIT evaluation. For the historical evaluation dataset, there were 12/1055 change tasks (i.e. queries) that were that committers first change (manually verified), and since there was no prior history for that developer we removed these from consideration for both changeset-batch and changeset-historical DIT.

There is an improvement in favor of historical simulation in MRR for only 1 of the 6 systems, while the remainder of the results are strongly in favor of batch changesets. Overall, batch changesets performs better than a full historical simulation, which suggests that under historical simulation, the accuracy of the DIT will fluctuate as a project evolves, which may indicate a more accurate evaluation is only possible with a historical simulation. Though the effect is smaller for DIT, this result aligns with the result we see for FLT that the two evaluations differ.

4.5 Configuration Analysis (RQ4)

A typical changeset corpus can be decomposed into four elements: additions (A) context lines (C), commit messages (M), and removals (R). Figure 1 shows each of these four elements. Intuitively, it is necessary to always include additions, as they contain the new words (code) that represent the feature that was being worked on at that moment. Likewise, the commit message is also an intuitive inclusion as it contains words that describe the change itself in natural language, which may help the model learn words more likely to be used in queries. It is less clear, however, whether to include both context lines and removals during corpus construction as they would increase noise of certain words, e.g., over emphasizing words that have already appeared in prior additions, for context lines (C), and duplicating value of words that have already appeared in additions when they are no longer valid, for removed words (R).

To gain insight into whether a particular source is beneficial, or detrimental, to performance of a task, we compare MRRs of all the possible configurations choices in Table 4. The table summarizes all configurations and their MRRs for each task of all subject systems and is computed using the snapshot corpus. We observe that the optimal configurations is (A, C, M) for FLT, resulting in a significant improvement in MRR relative to most other configurations. The optimal configuration for the DIT is (C, M) with the use of only the commit context lines (C) as a close second. The MRR for DIT exhibited less sensitivity to different configurations than for FLT with most of the MRR values in the range of

Subject System	Queries	MRR Snapshot	MRR Changesets	Spread	Wilcoxon p-value	Effect size
BookKeeper v4.3.0	164	**0.6600**	0.6513	−0.0086	$p = 0.0134$	−0.0632
Mahout v0.10.0	133	0.2374	**0.3340**	+0.0966	$p = 0.0606$	0.1884
OpenJPA v2.3.0	137	0.2728	**0.3648**	+0.0920	$p < 0.01$	0.2135
Pig v0.14.0	222	**0.2870**	0.1759	−0.1110	$p < 0.01$	−0.3536
Tika v1.8	40	**0.3915**	0.3609	−0.0306	$p = 0.3119$	−0.1194
ZooKeeper v3.5.0	359	**0.4542**	0.3985	−0.0557	$p = 0.6817$	−0.0547
All	1055	**0.3977**	0.3770	−0.0207	$p < 0.01$	−0.0412

(a) Snapshot-based vs. Changeset-Based Developer Identification: MRR, Wilcoxon p-values, and effect size

Subject System	Queries	MRR C-set Batch	MRR C-set Hist.	Spread	Wilcoxon p-value	Effect size
BookKeeper v4.3.0	163	**0.6522**	0.5871	−0.0651	$p = 0.7327$	−0.0451
Mahout v0.10.0	130	**0.3310**	0.2575	−0.0735	$p = 0.0273$	−0.1759
OpenJPA v2.3.0	136	**0.3650**	0.2757	−0.0892	$p = 0.0184$	−0.1603
Pig v0.14.0	221	0.1722	**0.2673**	+0.0950	$p < 0.01$	0.2883
Tika v1.8	39	0.3689	**0.4058**	+0.0369	$p = 0.1240$	0.1223
ZooKeeper v3.5.0	354	**0.3923**	0.3304	−0.0619	$p < 0.01$	−0.1752
All	1043	**0.3742**	0.3437	−0.0305	$p = 0.8541$	−0.0370

(b) Changeset-based Batch vs. Changeset-based Historical Developer Identification: MRR, Wilcoxon p-values, and effect size

TABLE 3
Results on Developer Identification.

TABLE 4
MRR values of all subject systems using different combinations of parts from the changesets (additions (A); context (C); messages (M); removals (R)).

Configuration	FLT	DIT
(A, R, C, M)	0.4315	0.3784
(A, R, C)	0.4092	0.3770
(A, R, M)	0.4119	0.3646
(A, R)	0.4214	0.3462
(A, C, M)	**0.4517**	0.3785
(A, C)	0.4010	0.3802
(A, M)	0.4147	0.3537
(A)	0.4031	0.3380
(R, C, M)	0.4013	0.3391
(R, C)	0.3443	0.3373
(R, M)	0.3488	0.3308
(R)	0.3151	0.3147
(C, M)	0.3971	**0.4165**
(C)	0.3386	0.4148
(M)	0.3838	0.3359

0.33 - 0.38. Note that for **RQ1**, **RQ2** and **RQ3**, we used the (A, R, C) configuration.

Examination of the results in Table 4 for both FLT and DIT indicates that, in most cases, it is beneficial to include additions, context, and messages, while excluding removals. Including removals reduces the MRR of both FLT and DIT in most configurations. This tends to match our intuitive view that removals would be detrimental to the performance of the topic model.

We examined the statistical significance of this trend using the Wilcoxon signed-rank test. The results for DIT are listed in Table 5 and for FLT in Table 6. For FLT, the results do not provide clear statistically significant evidence, except for (R, M) versus (M) where removals are shown to be detrimental. For DIT, the evidence is statistically significant most of the time and clearly points towards excluding removals from the configuration.

TABLE 5
Wilcoxon test results for removals inclusion and exclusion configurations of the DIT task for all subject systems.

Configurations		MRRs		p-value	Effect size
(A, R, C, M)	(A, C, M)	0.3784	**0.3785**	< 0.01	−0.0130
(A, R, C)	(A, C)	0.3770	**0.3802**	< 0.01	−0.0157
(A, R, M)	(A, M)	**0.3646**	0.3537	< 0.01	−0.0261
(A, R)	(A)	**0.3462**	0.3380	0.0596	−0.0111
(R, C, M)	(C, M)	0.3391	**0.4165**	< 0.01	0.1869
(R, C)	(C)	0.3373	**0.4148**	< 0.01	0.1591
(R, M)	(M)	0.3308	**0.3359**	< 0.01	0.0708

TABLE 6
Wilcoxon test results for removals inclusion and exclusion configurations of the FLT task for all subject systems.

Configurations		MRRs		p-value	Effect size
(A, R, C, M)	(A, C, M)	0.4315	**0.4517**	0.2007	0.0334
(A, R, C)	(A, C)	**0.4092**	0.4010	0.4550	−0.0096
(A, R, M)	(A, M)	0.4119	**0.4147**	0.0252	0.0167
(A, R)	(A)	**0.4214**	0.4031	0.2649	−0.0243
(R, C, M)	(C, M)	**0.4013**	0.3971	0.3120	0.0039
(R, C)	(C)	**0.3443**	0.3386	0.1753	0.0135
(R, M)	(M)	0.3488	**0.3838**	< 0.01	0.0908

4.6 Discussion

There is a strong performance advantage to using online LDA based on changesets rather than re-building the model from scratch using a snapshot. To characterize this benefit, Table 7 lists average model build times we experienced on

TABLE 7
Average time in seconds to train the LDA model and complete indexing for FLT task.

Subject System	Snapshot Indexing Time	Changeset Indexing Time
BookKeeper	214	79
Mahout	306	109
OpenJPA	795	54
Pig	1798	65
Tika	552	55
ZooKeeper	328	17

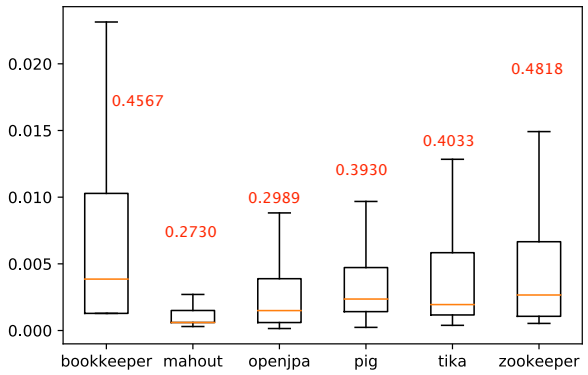
a commodity laptop computer for our set of study systems. The reduction in indexing cost, which includes training the LDA model and inferring the probability distribution of the terms in each file in order to construct the index, ranges from on the order of 2x to 20x, depending on the subject system. Since FLT and DIT models should be recomputed whenever a new maintenance issue is taken up by a developer, this cost can quickly add up in the rapidly changing projects that are now common in modern software development.

In addition to the performance advantage, our results show that there is also an accuracy advantage to using changeset-based topic modeling for some software projects and a disadvantage to using it in others. This accuracy variance is present among the two application domains we considered, feature location and developer identification. In this section, we aim to further unravel the characteristics of software projects that are likely to yield more success with changeset-based modeling.

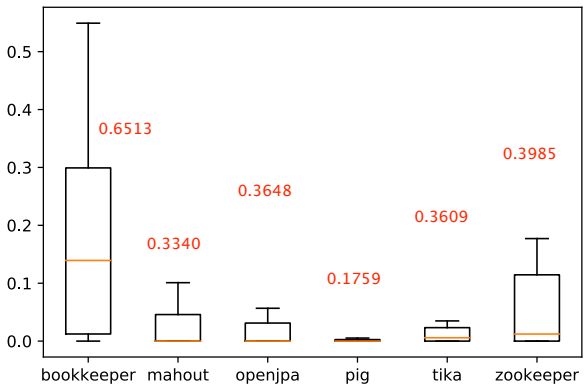
The major difference between changeset-based and snapshot-based models is that, in learning the model, for changeset-based the documents are the changesets, while for snapshot-based, the documents are structural program elements. Intuitively, an LDA model prefers terms and documents it has seen more often during training. Therefore, since the changeset-based topic model is built from changesets as documents, it should do better for queries on frequently changed parts of the source code. On the other hand, the snapshot-based model does not contain a bias towards frequently changed source code.

To evaluate this explanation for the accuracy of the models, we examined the change proneness [52] of the program elements in the FLT goldset of each of the software systems we studied. We compute change proneness as the average of the ratio of the number of changesets with a particular file (in the goldset) to the overall number of changesets in the project's training set. For DIT, a similar metric is the change involvement of each developer in the goldset, computed as the average of the ratio of the number of changesets contributed by a developer (mentioned in the goldset) to the overall number of changesets in the training set.

We show the results for FLT in Figure 5a and for DIT in Figure 5b. We observe that the MRR correlates with the change proneness and change involvement in the goldsets for FLT and DIT, respectively. That is, low values for these measures lead to lower MRR scores for systems and high values for these metrics lead to higher MRR scores. For instance, by far the lowest MRR for FLT trained on changesets was observed for Mahout, which also had the lowest



(a) Change proneness of the FLT goldset files in each project.



(b) Change involvement of each user in the DIT goldset.

Fig. 5. Change proneness (for FLT) and change involvement for (DIT) for each project; MRR scores from the prior evaluation are overlaid.

median change proneness. Similarly, for DIT, Pig produced the lowest MRR and the lowest change involvement among all of the studied systems. The Spearman test indicates that these correlations are strong and significant, with correlation coefficients of 0.89 for FLT MRR and change proneness and of 0.82 for DIT MRR and change involvement (both with $p < 0.05$).

The implications of this are that changeset-based topic models are more appropriate for systems where change is more localized and frequent in a particular part of the code, and poorer when change is highly distributed in the software project (i.e. with high entropy). In other words, when the model is used to retrieve frequently changed features in the code, it is very likely to recognize them and award them high scores. Conversely, the snapshot-based topic model is more uniform with respect to change, as the model simply reflects the distribution of terms across the structure of the code. Frequently changed systems are also those that are likely to require faster FLT or DIT model updating, where the performance advantage of the changeset based approach is clear.

5 Threats to Validity

Our studies have limitations that impact the validity of our findings, as well as our ability to generalize them. We describe some of these limitations and their impacts.

Construct Validity. Threats to construct validity concern measurements accurately reflecting the features of interest. A possible threat to construct validity is our benchmarks. Errors in the datasets could result in inaccurate effectiveness measures. Our dataset creation technique closely follows that of other researchers [43], [53], [54]. Additionally, datasets extracted source code entities automatically from changesets, previous work shows this approach is on par with manual extraction.

Another possible threat to construct validity is that we did not give special consideration to either tangled commits [55] (where two or more issues are resolved by a single commit) or multicommit resolutions [56] (where two or more commits resolve a single issue). Treating all files touched by a single commit as the goldset for multiple issues can potentially inflate the reported MRR values. However, the primary concern of this paper is to compare the accuracy of the changeset-based method to that of the snapshot-based method, and our treatment of tangled commits does not advantage (or disadvantage) either method. Conversely, treating multiple commits that address a single issue as separate changes can potentially decrease the reported MRR values (i.e., provide a conservative estimate of real-world accuracy).

Internal Validity. Threats to internal validity include possible defects in our tool chain and possible errors in our execution of the study procedure, the presence of which might affect the accuracy of our results and the conclusions we draw from them. We controlled for these threats by testing our tool chain and by assessing the quality of our data. Because we applied the same tool chain to all subject systems, any errors are systematic and are unlikely to affect our results substantially.

Another threat to internal validity pertains to the value of parameters such as the number of topics that we selected in training the models. We decided that the snapshot- and changeset-based approaches should have the same parameters to help facilitate evaluation and comparison. We argue that our study is not about selecting the best parameters, but to show that our snapshot-based approach is reasonable.

Further, since LDA is a stochastic model, we have determined a certain threat with respect to randomness in model construction. We control for this by ensuring each model created uses the same initial state. This is achieved by running each experiment in isolation and using a uniform random seed value on the system's pseudo-random number generator.

External Validity. Threats to external validity concern the extent to which we can generalize our results. The subjects of our study comprise six open source projects in Java, so we may not be able to generalize our results to systems implemented in other languages. However, the systems are of different sizes, are from different domains, and have characteristics in common with those of systems developed in industry.

Conclusion Validity. Threats to conclusion validity concern our choice of measurements and how those choices impact our evaluation and conclusion. We chose to use mean reciprocal rank (MRR), but we could have also used mean average precision (MAP) instead. We chose the former because it lends itself to being paired with the Wilcoxon signed-rank test as both rely on the same input data.

6 Conclusions

This paper analyzes the use of streaming (online) topic models built from changesets, generated from development activity of a software project, for text retrieval applications in software maintenance. As case studies, we examined feature location and developer identification, two well-studied and popular areas in software maintenance research. Using historical changes of several representative open source projects, we examined the effectiveness of changeset-based topic models, specific configuration options in training these models, and the effect of how different evaluation strategies account for time.

The results of our study indicate that: (1) online topic models are a suitable alternative to conventional topic models, (2) changeset-based topic models should use the additions, context and commit parts of the changeset, but avoid removals, (3) software projects with localized and frequent changes are likely to benefit the most from such models, and (4) the result of all feature location and developer identification experiments in this domain does not take time and software change into proper account (makes a *time invariance assumption*). Our future work is in experimenting with hybrid models that use a combination of snapshots and changesets, leveraging online topic models for additional software maintenance problems, and in improving evaluation of feature location technique with respect to the time invariance assumption.

References

[1] A. Marcus, A. Sergeyev, V. Rajlich, and J. I. Maletic, "An information retrieval approach to concept location in source code," in *Proceedings of the 11th Working Conference on Reverse Engineering*. IEEE, 2004, pp. 214–223. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1374321

[2] H. Sajnani, V. Saini, J. Svajlenko, C. K. Roy, and C. V. Lopes, "Sourcerercc: Scaling code clone detection to big-code," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, May 2016, pp. 1157–1168.

[3] A. Abadi, M. Nisenson, and Y. Simionovici, "A traceability technique for specifications," in *Program Comprehension, 2008. ICPC 2008. The 16th IEEE International Conference on*, June 2008, pp. 103–112.

[4] S. K. Lukins, N. A. Kraft, and L. H. Etzkorn, "Source code retrieval for bug localization using latent dirichlet allocation," in *Reverse Engineering, 2008. WCRE'08. 15th Working Conference on*. IEEE, 2008, pp. 155–164. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4656405

[5] L. R. Biggers, "Investigating the effect of corpus construction on latent dirichlet allocation based feature location," Ph.D. dissertation, University of Alabama, 2012.

[6] K. Damevski, H. Chen, D. Shepherd, N. Kraft, and L. Pollock, "Predicting future developer behavior in the IDE using topic models," *IEEE Transactions on Software Engineering*, 2017.

[7] M. Hoffman, F. R. Bach, and D. M. Blei, "Online learning for latent dirichlet allocation," in *Advances in Neural Information Processing Systems 23*, J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, Eds. Curran Associates, Inc., 2010, pp. 856–864. [Online]. Available: http://papers.nips.cc/paper/3902-online-learning-for-latent-dirichlet-allocation.pdf

[8] C. S. Corley, K. L. Kashuda, and N. A. Kraft, "Modeling changeset topics for feature location," in *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2015. [Online]. Available: http://christop.club/publications/pdfs/Corley-etal_2015.pdf

[9] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *the Journal of machine Learning research*, vol. 3, pp. 993–1022, 2003. [Online]. Available: http://dl.acm.org/citation.cfm?id=944937

[10] V. Rajlich and N. Wilde, "The role of concepts in program comprehension," in *Program Comprehension, 2002. Proceedings. 10th International Workshop on*. IEEE, 2002, pp. 271–278. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1021348

[11] K. Damevski, D. Shepherd, and L. Pollock, "A field study of how developers locate features in source code," *Empirical Software Engineering*, vol. 21, no. 2, pp. 724–747, Apr 2016.

[12] V. Singh, W. Snipes, and N. Kraft, "A framework for estimating interest on technical debt by monitoring developer activity related to code comprehension," in *Proc. 6th IEEE Int'l Wksp. on Managing Technical Debt*, ser. MTD'14, 2014.

[13] D. Chatterji, J. Carver, N. Kraft, and J. Harder, "Effects of cloned code on software maintainability: A replicated developer study," in *Proc. 20th Working Conf. on Reverse Engineering*, ser. WCRE'13, 2013, pp. 112–121.

[14] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman, "Indexing by latent semantic analysis," *Journal of the American Society of Information Science*, vol. 41, no. 6, pp. 391–407, 1990. [Online]. Available: http://www.cob.unt.edu/itds/faculty/evangelopoulos/dsci5910/LSA_Deerwester1990.pdf

[15] D. Poshyvanyk and A. Marcus, "Combining formal concept analysis with information retrieval for concept location in source code," in *15th IEEE International Conference on Program Comprehension (ICPC '07)*. Washington, DC, USA: IEEE, Jun. 2007, pp. 37–48.

[16] D. Liu, A. Marcus, D. Poshyvanyk, and V. Rajlich, "Feature location via information retrieval based filtering of a single scenario execution trace," in *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*. ACM, 2007, pp. 234–243. [Online]. Available: http://dl.acm.org/citation.cfm?id=1321667

[17] G. Scanniello and A. Marcus, "Clustering support for static concept location in source code," in *Program Comprehension (ICPC), 2011 IEEE 19th International Conference on*. IEEE, 2011, pp. 1–10. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5970158

[18] M. Eaddy, A. V. Aho, G. Antoniol, and Y.-G. Guéhéneuc, "Cerberus: Tracing requirements to source code using information retrieval, dynamic analysis, and program analysis," in *Program Comprehension, 2008. ICPC 2008. The 16th IEEE International Conference on*. IEEE, 2008, pp. 53–62. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4556117

[19] D. Cubranic, G. C. Murphy, J. Singer, and K. S. Booth, "Hipikat: A project memory for software development," *Software Engineering, IEEE Transactions on*, vol. 31, no. 6, pp. 446–465, 2005. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1463229

[20] D. Poshyvanyk, Y.-G. Guéhéneuc, A. Marcus, G. Antoniol, and V. Rajlich, "Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval," *Software Engineering, IEEE Transactions on*, vol. 33, no. 6, pp. 420–432, 2007. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4181710

[21] S. K. Lukins, N. A. Kraft, and L. H. Etzkorn, "Bug localization using latent dirichlet allocation," *Information and Software Technology*, vol. 52, no. 9, pp. 972–990, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584910000650

[22] L. R. Biggers, C. Bocovich, R. Capshaw, B. P. Eddy, L. H. Etzkorn, and N. A. Kraft, "Configuring latent dirichlet allocation based feature location," *Empirical Software Engineering*, vol. 19, no. 3, pp. 465–500, 2014.

[23] B. Dit, L. Guerrouj, D. Poshyvanyk, and G. Antoniol, "Can better identifier splitting techniques help feature location?" in *2011 IEEE 19th International Conference on Program Comprehension*. IEEE, Jun 2011. [Online]. Available: http://dx.doi.org/10.1109/ICPC.2011.47

[24] C. S. Corley, K. Damevski, and N. A. Kraft, "Exploring the use of deep learning for feature location," in *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2015. [Online]. Available: http://christop.club/publications/pdfs/Corley-etal_2015a.pdf

[25] B. Bassett and N. A. Kraft, "Structural information based term weighting in text retrieval for feature location," in *Program Comprehension (ICPC), 2013 IEEE 21st International Conference on*. IEEE, 2013, pp. 133–141. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6613841

[26] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Inf. Process. Manage.*, vol. 24, no. 5, pp. 513–523, 1988.

[27] R. K. Saha, M. Lease, S. Khurshid, and D. E. Perry, "Improving bug localization using structured information retrieval," in *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*. Institute of Electrical & Electronics Engineers (IEEE), Nov 2013. [Online]. Available: http://dx.doi.org/10.1109/ASE.2013.6693093

[28] S. Rao, H. Medeiros, and A. Kak, "An incremental update framework for efficient retrieval from software libraries for bug localization," in *Reverse Engineering (WCRE), 2013 20th Working Conference on*. IEEE, 2013, pp. 62–71. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6671281

[29] D. W. McDonald and M. S. Ackerman, "Just talk to me: a field study of expertise location," in *Proceedings of the 1998 ACM conference on Computer supported cooperative work*, ser. CSCW '98. New York, NY, USA: ACM, 1998, pp. 315–324.

[30] A. Begel, Y. P. Khoo, and T. Zimmermann, "Codebook: discovering and exploiting relationships in software repositories," in *Software Engineering, 2010 ACM/IEEE 32nd International Conference on*, vol. 1. IEEE, 2010, pp. 125–134.

[31] R. Shokripour, J. Anvik, Z. M. Kasirun, and S. Zamani, "Why so complicated? simple term filtering and weighting for location-based bug report assignment recommendation," in *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, May 2013. [Online]. Available: http://dx.doi.org/10.1109/MSR.2013.6623997

[32] D. W. McDonald and M. S. Ackerman, "Expertise recommender: a flexible recommendation system and architecture," in *Proceedings of the 2000 ACM conference on Computer supported cooperative work*. ACM, 2000, pp. 231–240. [Online]. Available: http://dl.acm.org/citation.cfm?id=358994

[33] S. Minto and G. C. Murphy, "Recommending emergent teams," in *Fourth International Workshop on Mining Software Repositories (MSR'07:ICSE Workshops 2007)*. Washington, DC, USA: IEEE, May 2007, p. 5.

[34] H. Kagdi, M. Hammad, and J. I. Maletic, "Who can help me with this source code change?" in *2008 IEEE International Conference on Software Maintenance*. IEEE, Sep. 2008, pp. 157–166.

[35] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu, "Don't touch my code!: examining the effects of ownership on software quality," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. ACM, 2011, pp. 4–14. [Online]. Available: http://dl.acm.org/citation.cfm?id=2025119

[36] M. Linares-Vásquez, K. Hossen, H. Dang, H. Kagdi, M. Gethers, and D. Poshyvanyk, "Triaging incoming change requests: Bug or commit history, or code authorship?" in *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*. IEEE, 2012, pp. 451–460. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6405306

[37] M. K. Hossen, H. Kagdi, and D. Poshyvanyk, "Amalgamating source code authors, maintainers, and change proneness to triage change requests," in *Proceedings of the 22nd International Conference on Program Comprehension*. ACM Press, 2014. [Online]. Available: http://dx.doi.org/10.1145/2597008.2597147

[38] A. Tamrawi, T. T. Nguyen, J. M. Al-Kofahi, and T. N. Nguyen, "Fuzzy set and cache-based approach for bug triaging," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. ACM, 2011, pp. 365–375.

[39] M. F. Porter, "An algorithm for suffix stripping," *Program: electronic library and information systems*, vol. 14, no. 3, pp. 130–137, 1980. [Online]. Available: http://www.emeraldinsight.com/journals.htm?articleid=1670983&show=abstract

[40] R. Vasa, J.-G. Schneider, and O. Nierstrasz, "The inevitable stability of software change," in *Software Maintenance, 2007. ICSM 2007. IEEE International Conference on*. IEEE, 2007, pp. 4–13.

[41] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl, "Mining version histories to guide software changes," *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 429–445, 2005.

[42] C. S. Corley, K. L. Kashuda, D. S. May, and N. A. Kraft, "Modeling changeset topics," in *Proc. 4th Wksp. on Mining Unstructured*

Data, 2014. [Online]. Available: http://cscorley.students.cs.ua.edu/publications/pdfs/Corley-etal_14.pdf

[43] L. Moreno, J. J. Treadway, A. Marcus, and W. Shen, "On the use of stack traces to improve text retrieval-based bug localization," in *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*, 2014, pp. 151–160. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6976081

[44] H. Kagdi, M. Gethers, D. Poshyvanyk, and M. Hammad, "Assigning change requests to software developers," *J. Softw. Evol. and Proc.*, vol. 24, no. 1, pp. 3–33, 2012.

[45] T. Bissyande, F. Thung, S. Wang, D. Lo, L. Jiang, and L. Reveillere, "Empirical evaluation of bug linking," in *Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on*, March 2013, pp. 89–98.

[46] G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with bug tossing graphs," in *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ser. ESEC/FSE '09. New York, NY, USA: ACM, 2009, pp. 111–120.

[47] P. Bhattacharya, I. Neamtiu, and C. R. Shelton, "Automated, highly-accurate, bug assignment using machine learning and tossing graphs," *Journal of Systems and Software*, vol. 85, no. 10, p. 22752292, Oct 2012. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2012.04.053

[48] B. Croft, D. Metzler, and T. Strohman, *Search engines : information retrieval in practice*. Boston: Addison-Wesley, 2010. [Online]. Available: http://www.search-engines-book.com/

[49] N. Cliff, "Dominance statistics: ordinal analyses to answer ordinal questions," *Psychological Bulletin*, vol. 144, no. 3, pp. 494–509, Nov. 1993.

[50] J. Romano, J. D. Kromrey, J. Coraggio, and J. Skowronek, "Appropriate statistics for ordinal level data: Should we really be using t-test and cohensd for evaluating group differences on the nsse and other surveys," in *Annual Meeting of the Florida Association of Institutional Research*, 2006, pp. 1–33.

[51] D. Matter, A. Kuhn, and O. Nierstrasz, "Assigning bug reports using a vocabulary-based expertise model of developers," in *2009 6th IEEE International Working Conference on Mining Software Repositories*. Washington, DC, USA: IEEE, May 2009, pp. 131–140.

[52] J. M. Bieman, A. A. Andrews, and H. J. Yang, "Understanding change-proneness in oo software through visualization," in *11th IEEE International Workshop on Program Comprehension, 2003.*, May 2003, pp. 44–53.

[53] B. Dit, A. Holtzhauer, D. Poshyvanyk, and H. Kagdi, "A dataset from change history to support evaluation of software maintenance tasks," in *Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on*. IEEE, 2013, pp. 131–134. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6624019

[54] M. Revelle, B. Dit, and D. Poshyvanyk, "Using data fusion and web mining to support feature location in software," in *Program Comprehension (ICPC), 2010 IEEE 18th International Conference on*. IEEE, 2010, pp. 14–23. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5521780

[55] K. Herzig and A. Zeller, "The impact of tangled code changes," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, May 2013, pp. 121–130.

[56] S. McIntosh, B. Adams, T. Nguyen, Y. Kamei, and A. Hassan, "An empirical study of build maintenance effort," in *Proceedings of the International Conference on Software Engineering*, 2011, pp. 141–150.

Christopher S. Corley is a software engineer based in Chattanooga, Tennessee. He received the Ph.D. degree in computer science from The University of Alabama in 2018. His research interests are in software engineering and maintenance, specifically applications of topic modeling to automation of software maintenance tasks.

Kostadin Damevski is an Assistant Professor at the Department of Computer Science at Virginia Commonwealth University. Prior to that he was a faculty member at the Department of Computer Science at Virginia State University and a postdoctoral research assistant at the Scientific Computing and Imaging institute at the University of Utah. His research focuses on information retrieval techniques and recommendation systems for software maintenance. Dr. Damevski received a Ph.D. in Computer Science from the University of Utah in Salt Lake City.

Nicholas A. Kraft is a software researcher at ABB Corporate Research in Raleigh, North Carolina. Previously, he was an associate professor in the Department of Computer Science at The University of Alabama. He received the Ph.D. degree in computer science from Clemson University in 2007. His research interests are in software evolution, with an emphasis on techniques and tools to support developers in understanding evolving software and to support managers in understanding software evolution processes. Dr. Kraft's research has been funded by grants from the NSF, DARPA, and ED. He currently serves on the editorial board of IEEE Software and on the steering committee of the IEEE International Conference on Software Maintenance and Evolution (ICSME). He is a senior member of the ACM and the IEEE.