ONLINE TOPIC MODELING FOR SOFTWARE MAINTENANCE USING A CHANGESET-BASED APPROACH

by

CHRISTOPHER SCOTT CORLEY

JEFFREY C. CARVER, COMMITTEE CHAIR
JEFFREY G. GRAY
NICHOLAS A. KRAFT
RANDY K. SMITH
TRAVIS L. ATKISON

A DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
in the Graduate School of
The University of Alabama

TUSCALOOSA, ALABAMA

2018

ABSTRACT

Topic modeling is a machine learning technique for discovering thematic structure within a corpus. Topic models have been applied to several areas of software engineering, including bug localization, feature location, triaging change requests, and traceability link recovery. Many of these approaches train topic models on a source code snapshot – a revision or state of code at a particular point of time, such as a versioned release. However, source code evolution leads to model obsolescence and thus to the need to retrain the model from the latest snapshot, incurring a non-trivial computational cost of model re-learning.

This work proposes and investigates an approach that can remedy the obsolescence problem. Conventional wisdom in the software maintenance research community holds that the topic model training information must be the same information that is of interest for retrieval. The primary insight for this work is that topic models can infer the topics of any information, regardless of the information used to train the model. Pairing online topic modeling with mining software repositories, I can remove the need to retrain a model and achieve model persistence. For this, I suggest training of topic models on the software repository history in the form of the *changeset* – a textual representation of the changes that occur between two source code snapshots.

To show the feasibility of this approach, I investigate two popular applications of text retrieval in software maintenance, feature location and developer identification. Feature location is a search activity for locating the source code entity that relates to a feature of interest. Developer identification is similar, but focuses on identifying the developer most apt for working on a feature

of interest. Further, to demonstrate the usability of changeset-based topic models, I investigate whether I can coalesce topic-modeling-based maintenance tasks into using a *single* model, rather than needing to train a model for each task at hand. In sum, this work aims to show that *training online topic models on software repositories removes retraining costs while maintaining accuracy of a traditional snapshot-based topic model for different software maintenance problems.*

DEDICATION

For Mom and Sis

LIST OF ABBREVIATIONS AND SYMBOLS

α	in LDA, a smoothing parameter for document-topic distributions
η	in LDA, a smoothing parameter for topic-term distributions
θ	the document-topic matrix
ϕ	the term-topic matrix
d	a document
q	a query, or a document of interest
C	a corpus, or the term-document matrix
V	the vocabulary of a corpus
w	a term
z	a topic
K	in LDA, the number of topics; in LSI, the reduction factor
M	the number of unique terms in a corpus
N	the number of documents in a corpus
S	In LSI, a singular values matrix
BLT	Bug Localization Technique
DIT	Developer Identification Technique
FLT	Feature Location Technique

IR	Information Retrieval
LDA	Latent Dirichlet Allocation
LOC	Lines of Code
LSA	Latent Semantic Analysis
LSI	Latent Semantic Indexing
PLSI	Probabilistic Latent Semantic Indexing
SLOC	Source Lines of Code
SVD	Singular Value Decomposition
VCS	Version Control System
VSM	Vector Space Model

ACKNOWLEDGMENTS

There are many, *many* people I would like to acknowledge and thank for their continued support and friendship over the years. Honestly, I have too many people I would like to thank and I am sorry that I will not be able to thank everyone.

I will start with the obvious and thank my advisor, Nicholas Kraft. I thank him for taking a risk on me and allowing me to study under him. I would also like to thank him for always being there when I needed support, for being patient and understanding, and most importantly for encouraging me to be a better person and to do more with my life. He has always been, and always will be, more than just an advisor to me – he is a friend.

Of course, this dissertation would not have materialized without the guidance and encouragement of my committee. I would especially like to thank Dr. Carver for becoming my committee chair in the middle of my studies and keeping me on track to graduate.

I thank my various co-authors, to everyone that had the patience to work with me, and to those that encouraged me to continue this work. I'd particularly like to thank Hazel Victoria Campbell and Kelly Kashuda. Other co-authors include those I worked with during my year at ABB Corporate Research. Thank you all for allowing me to do something different, and for helping me manually solder a bunch of silly lights! Thank you David Shepherd for allowing me to pursue such a fun project and for encouragement to finish this dissertation.

Thank you to the software engineering research community for their encouragement, support,

and kind wisdom. I'd like to thank Chris Parnin, Abram Hindle, Felienne Hermans, Mike Godfrey, Neil Ernst, Patrick Wagstrom, and Dave Binkley.

I would also not be here without the emotional support and friendship I found during my time in Tuscaloosa. Thank you Heather Wyatt, Brian Oliu, Bob Weatherly, Chris Skinner, Austin Stickney, Drew Brooks, Erik Kline, and Jeb Richter. I always look forward to seeing you whenever I visit Tuscaloosa.

I would also like to thank those I've befriended in Chattanooga. Thank you in particular to Brian Hennen and Amanda Norris for being great friends that I look forward to knowing for a long time to come. Thanks also to my former and current coworkers Steve Medlin, Russ Wilson, and James Gardner for their patience, understanding, and encouragement as I attempted to find a balance between working and researching.

Thanks to my all my lifelong friends for always being there for me. Thank you Cliff and Clay Davis for their friendship, support, encouragement, and wisdom. Thanks to David Stewart and the rest of the Stewart family for getting me through my undergraduate studies and into graduate school. My final thank you goes to my dear friend, James Holden, for being my longest and kindest friend, and for sparking my interest in computing.

I have been so fortunate to know all of you.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

Software features are functionalities defined by requirements and accessible to developers and users. Software change is continual, because revised requirements lead to new features, increasing expectations lead to feature enhancements, and achieving intended behavior leads to removal of defective features (i.e., bugs). This software maintenance and evolution is driven by users, project managers, and developers requesting changes. Specific kinds of change requests include feature requests, enhancement requests, and bug reports. Each change request requiring a code change is assigned to a developer to implement.

A developer assigned to a change request must first understand what a program does and how the program is implemented, i.e., program comprehension is a prerequisite to incremental change [Corbi, 1989]. For example, the developer spends effort to understand the system architecture or to locate the parts of the source code implementing the feature(s). For developers who are unfamiliar with the system, gaining such knowledge can be a time-consuming task [Müller, Jahnke, Smith, Storey, Tilley, and Wong, 2000]. One approach to assisting building program comprehension is to use text retrieval techniques [Lukins, Kraft, and Etzkorn, 2008].

Developer identification is the process of determining which developer has appropriate expertise [McDonald and Ackerman, 1998]. That is, it is the process of finding a developer that already has the prerequisite knowledge of the software system. Indeed, developers need help finding expertise within their organization *more than they need help finding source code elements* [Begel,

Khoo, and Zimmermann, 2010]. Unfortunately, developer identification can be an error-prone and time-consuming process when completed manually. Jeong, Kim, and Zimmermann [2009] found that change request reassignment occurs between 37%-44% of the time and introduces an average delay of 50 days until the request is completed. It is imperative that a request is assigned to the best-fit developer the first time. There are a myriad of approaches to this problem [Shokripour, Anvik, Kasirun, and Zamani, 2013], many of which are based, at least in part, on text retrieval techniques [Kagdi, Gethers, Poshyvanyk, and Hammad, 2012].

One text retrieval technique is topic modeling. In software engineering, topic models uncover thematic structure (e.g., topics) of source code entities grouped by their natural language content (i.e., the words in their identifiers, comments, and literals). Such topics often correspond to the concepts and features implemented by the source code [Baldi, Lopes, Linstead, and Bajracharya, 2008], and exploring such topics shows promise in helping developers to understand the entities that make up a system and to understand how those entities relate [Gethers, Savage, Di Penta, Oliveto, Poshyvanyk, and De Lucia, 2011; Kuhn, Ducasse, and Gírba, 2007; Maskeri, Sarkar, and Heafield, 2008; Savage, Dit, Gethers, and Poshyvanyk, 2010]. Recent approaches to exploring linguistic topics in source code use machine learning techniques that model correlations among words, such as latent semantic indexing (LSI) [Deerwester, Dumais, Landauer, Furnas, and Harshman, 1990] and latent Dirichlet allocation (LDA) [Blei, Ng, and Jordan, 2003], and machine learning techniques that also model correlations among documents, such as Relational Topic Models [Chang and Blei, 2010].

Useful for more than just general program comprehension, topic models of source code have concrete applications including: feature location [Dit, Revelle, Gethers, and Poshyvanyk, 2013],

bug localization [Lukins et al., 2008; Rao, Medeiros, and Kak, 2013], developer identification [Kagdi et al., 2012], traceability link recovery [Asuncion, Asuncion, and Taylor, 2010], and other areas [Biggers, 2012]. Yet, while researchers have had success in using topic models on source code entities for a variety of applications, a fundamental issue exists with the current approaches: topic modeling algorithms typically assume the input is comprised of immutable documents, while source code entities are mutable. This conflict in assumptions is the motivating point of this work.

1.1 Terminology

This section defines some of the terminology seen throughout this work. I adopt and extend terminology from Biggers, Bocovich, Capshaw, Eddy, Etzkorn, and Kraft [2014]. In particular, I define the following:

feature location the act of identifying the source code entity or entities implementing a feature

developer identification the act of identifying the developer most apt to completing a software maintenance task

query, q a document created by a user

term (word), w the smallest free-form of a language

token a sequence of non-whitespace characters containing one or more term

entity a named source element such as a method, class, or package

identifier a token representing the name of an entity

comment a sequence of tokens delimited by language-specific markers (e.g., `/* */` or `#`)

literal a sequence of tokens delimited by language-specific markers (e.g., `' '` for strings)

document, d a sequence of D terms $w_1, w_2, ..., w_D$, often represented as a bag of words (i.e., M-length vector of term frequencies or weights)

corpus, C a sequence of N documents $d_1, d_2, ..., d_N$ (i.e., a $M \times N$ term-document matrix)

vocabulary, V a set of M unique terms that appear in a corpus $\{w_1, w_2, ..., w_M\}$

topic, z a concept or theme of related terms, often represented as a distribution of term proportions

topic model, ϕ a mathematical representation of the thematic structure of a corpus, or how much a term w contributes to each topic z (i.e., a $K \times M$ topic-term matrix)

inferrence, θ_d the thematic structure of a given document (i.e., a K-length topic proportion vector)

index, θ a structure optimized for searching, e.g., in LDA by inferring the thematic structure of each document (i.e., a $N \times K$ document-topic matrix)

search engine ranks documents by their similarity to a query

diff set of text which represents the differences between two texts (see Figure 1.1)

patch a set of instructions (i.e., diffs) used to transform one set of texts into another

context lines lines of a diff that denote text useful for transforming the text, but do not represent the differences

added lines lines of a diff that were *added* to transform the first text into the second

removed lines lines of a diff that were *removed* to transform the first text into the second

changeset ideally represents a single feature modification, addition, or deletion, which may crosscut two or more source code entities

version control system (VCS) tool that assists a developer in maintaining software , such as Git or Subversion

commit a representation of a changeset in a version control system

software repository collection of commits over time which represent the history of a software project, maintained by a VCS

```
commit b1432f097ada17573c2dbf81e982915e3e81c815
Author: Tim Allison <tallison@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;
                 }
             }
         });
```

Figure 1.1: Example of a `git diff`
This changeset addresses Tika v1.8 Issue #1689. 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 addition lines, and red lines (beginning with a single −) denote removals 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.

1.2 Motivation

When modeling a source code repository, the corpus typically represents a snapshot of the code. That is, a topic model is often trained on a corpus that contains documents that represent source code entities (e.g., files, classes, or methods) from a particular version of the software at a particular moment in time. Keeping such a model up-to-date is expensive, because the frequency and scope of source code changes necessitate retraining the model on the updated corpus [Rao, 2013]. It may be possible to automate certain maintenance tasks without a model of the complete source code. For example, when assigning a developer to a change task, I can use a topic model to relate developers with topics that characterize their previous changes. In this scenario, a model of

the text changed by each developer may be more useful than a model of the entities changed by each developer.

While using entity-based models is a natural fit for program comprehension tasks such as feature location and bug localization, they still are unable to stay up-to-date entirely [Rao, 2013]. Further, much of the work for assigning developers to change requests still use source code entities as input and an array of heuristics to identify a developer [Hossen, Kagdi, and Poshyvanyk, 2014; Kagdi et al., 2012]. These methods also have the same flaw in that they ultimately rely on source code entities for information.

To remedy these shortcomings, I propose to use *changesets* in the training of a topic model. Like Rao et al. [2013], the motivation of this work is to create topic models that can be incrementally updated over time. Unlike Rao et al. [2013], I can rely on the source code history itself to build the model without needing to manually adjust model latent variables to account for document changes. This gains the benefit of a decrease in model construction and query times, could lead to a more reliable model, and for software maintenance tasks removes the need for custom implementations of complex topic modeling algorithms.

The key intuition to this approach is that a topic model algorithm such as latent Dirichlet allocation can *infer* any given document's topic proportions regardless of the documents used to train the model. That is, I can train a model a corpus of changesets and infer the topics of an entirely different corpus (e.g., source code entities). Further, now that topic modeling algorithms are online [Hoffman, Bach, and Blei, 2010; Řehůřek, 2011], the model can be periodically updated with new changesets as they enter a source code repository. With this approach, the topic model is up-to-date with the current source code, even as developers are using the model for work.

Thesis Statement Training online topic models on software repositories removes retraining costs while maintaining accuracy of a traditional snapshot-based topic model for different software maintenance problems.

1.2.1 Feature Location

The state-of-the-practice in feature location is to use an IDE tool based on keyword or regular expression search, but Ko, Myers, Coblenz, and Aung [2006] observed such tools leading developers to failed searches nearly 90% of the time. The state-of-the-art in feature location [Dit et al., 2013] is to use a feature location technique (FLT) based, at least in part, on text retrieval (TR). The standard methodology [Marcus, Sergeyev, Rajlich, and Maletic, 2004] is to extract a document for each class or method in a source code snapshot, to train a TR model on those documents, and to create an index of the documents from the trained model. Topic models (TMs) [Blei, 2012] such as latent Dirichlet allocation (LDA) [Blei et al., 2003] are the state-of-the-art in TR and outperform vector-space models (VSMs) in the contexts of natural language [Blei et al., 2003; Deerwester et al., 1990] and source code [Lukins, Kraft, and Etzkorn, 2010; Poshyvanyk, Guéhéneuc, Marcus, Antoniol, and Rajlich, 2007]. Yet, modern TMs such as online LDA [Hoffman et al., 2010] natively support only the online addition of a new document, whereas some VSMs also natively support online modification or removal of an existing document. So, TM-based FLTs provide the best accuracy, but unlike VSM-based FLTs, they require computationally-expensive retraining subsequent to source code changes.

Rao [2013] proposed FLTs based on customizations of LDA and latent semantic indexing (LSI) that support online modification and removal. These FLTs require less-frequent retraining

than other TM-based FLTs, but the remaining cost of periodic retraining inhibits their application to large software, and the reliance on customization hinders their extension to new TMs.

I envision an FLT that is:

1) accurate like a TM-based FLT,
2) inexpensive to update like a VSM-based FLT,
3) and extensible to accommodate any off-the-shelf TR model that supports online addition of a new document.

Unfortunately, my vision is incompatible with the standard methodology for FLTs. Existing VSM-based FLTs fail to satisfy the first criteria, and existing TM-based FLTs fail to satisfy the second or third criteria. Indeed, given the current state-of-the-art in TR, it is impossible for a FLT to satisfy all three criteria while following the standard methodology.

In this work, I propose a new methodology for FLTs. My methodology is to extract a document for each changeset in the source code history and to train a TR model on the changeset documents, and then to extract a document for each source file in a source code snapshot and to create an index of the source file documents from the trained (changeset) model. This new methodology stems from three key observations:

1) like a source code file, a changeset contains program text,
2) unlike a source code file, a changeset is immutable,
3) and an atomic changeset involves a single feature.

It follows from the first two observations that it is possible for an FLT following my methodology to satisfy all three of the aforementioned criteria. The next two observations influence

the training and indexing steps of my methodology, which have the conceptual effect of relating source files to changeset topics. By contrast, the training and indexing steps of the standard methodology have the conceptual effect of relating files to file topics.

1.2.2 Developer Identification

The state-of-the-practice in developer identification is largely a manual, communicative process. Developer identification is a common and difficult task, and is even more difficult on projects where developer teams are large or geographically distributed [Herbsleb, Mockus, Finholt, and Grinter, 2001]. Developer identification requires contextual knowledge about the product, team structure, individual expertise, workload balance, and development schedules in order to correctly assign a change request. A project member triaging a change request will need to consider these factors in order to correctly assign the change request to a set of developers with appropriate expertise [McDonald and Ackerman, 1998].

The state-of-the-art in developer identification [Kagdi et al., 2012] aims to automate this process, either partially or fully. Anvik, Hiew, and Murphy [2006] notes that a fully-automated approach may not be feasible given the amount of contextual knowledge required for triage. One approach is to use a developer identification technique (DIT) based, again at least in part, on text retrieval (TR). This category of approaches, as described by Shokripour et al. [2013], involves combining an FLT to locate the features in source code and using change heuristics to select the correct developer [Bird, Nagappan, Murphy, Gall, and Devanbu, 2011; Corley, Kammer, and Kraft, 2012; Hossen et al., 2014]. Any DIT relying on a TR-based FLT will exhibit the same issues as I described in the previous section.

I would like to further adapt my vision for an accurate, easy-to-update FLT to a DIT. In

this work, I propose a new methodology for DITs that is not necessarily based on the performance of an FLT. Again, my methodology is to extract a document for each changeset in the source code history and to train a TR model on the changeset documents. However, rather than using a heuristic-and-FLT-based approach, I choose to follow Matter, Kuhn, and Nierstrasz [2009] and extract *developer profiles*: documents which describe all changes a developer has made on the project. I can then create an index of the developer profiles from the trained (changeset) model and skip all heuristics.

1.2.3 Combining and Configuring Changeset-based Topic Models

Topic model reuse for two tasks would halve the computational cost required for model training. This presumes, however, that the configuration choices made for corpus construction and the model itself are acceptable for both tasks. That is, the model used does not negatively impact either task due to configuration.

Biggers et al. [2014] were the first to explore the parameters of a LDA-based FLT. Unfortunately, these findings for model training parameters may not directly apply to a LDA-based DIT. Further, there is no work on optimal configurations when using a topic model for two tasks. There is also evidence that different configurations may be better suited for different tasks [Abadi, Nisenson, and Simionovici, 2008; Marcus and Poshyvanyk, 2005].

Likewise, I must also make choices with respect to corpus construction, as LDA can achieve higher performance by adjusting certain elements of a corpus [Biggers et al., 2014] or by increasing and decreasing the importance of certain elements of the corpus [Bassett and Kraft, 2013]. While I do not have enough source code in changesets to extract fully parsed elements – such as comments, identifiers, literals, and so on – I do have structure in the changeset itself in the form of the `diff` (see

Figure 1.1). That is, I have lines removed and added that represent the change, lines for context for where the change is to be applied, and a natural language commit message describing the change.

1.3 Research Goals and Problems

The primary research goal of this work is to evaluate the performance and reliability of topic models built on the source code histories, i.e., changeset-based topic modeling. This requires configuring and executing studies in different contexts of software maintenance work, such as feature location and developer identification. Towards achieving this goal, I've identified three core research problems, defined below.

1.3.1 Effectively using software repository history for an online topic-modeling-based feature location technique to remove retraining costs

I investigate the usefulness of using changesets to construct models with online LDA for feature location. Conventional feature location techniques train models using only the source code in the form of a snapshot of the source code. I explore how to train models on the changesets, and then use that model for inference of a source code snapshot.

1.3.2 Effectively using software repository history for an online topic-modeling-based developer identification technique to remove retraining costs

Like feature location, many developer identification techniques train models using only a source code snapshot. Developer identification diverges from feature location once a model is obtained, using that model to find an appropriate developer with a myriad of heuristics and techniques. I investigate the usefulness of using changesets to construct models with online LDA

for developer identification. I explore how to train models on the changesets, and then use that model for inference of an individual developer's word change history.

1.3.3 Effectively using a *single* changeset-based topic model for automating different software maintenance tasks

Here, I explore whether I can re-use the same model for different tasks. In particular, because topic models are sensitive to parameter selection [Biggers et al., 2014], I perform parameter sweeps across both tasks of feature location and developer identification. I seek, per project, an optimal parameter configuration that works well on both tasks. Further, as changesets themselves have different features, such as added, removed, and context lines, I investigate which seem to be most beneficial to each approach.

1.4 Organization

The organization of this work is as follows. Chapter 2 discusses related work in the area of text retrieval, configuration, and applications in software maintenance. Chapter 3 describes the methodologies of the changeset-based approach as they apply to each research problem. I present and discuss the results of the studies in chapters 4 and 5, respectively. Finally, in Chapter 6 I conclude and describe future directions for this work.

2. BACKGROUND & RELATED WORK

In this section, I review the literature on text retrieval (TR) approaches and applying TR approaches for software maintenance tasks. In particular, I discuss TR applications in feature location and developer identification. Finally, I cover the related works of feature location and developer identification.

2.1 Text Retrieval

In this section, I review and summarize the text retrieval process, which consists of two general steps: document extraction and document retrieval. I discuss methods of measuring similarity. Finally, I discuss measures for evaluating a text retrieval technique.

2.1.1 Document Extraction

The left side of Figure 2.1 illustrates the document extraction process. A document extractor takes raw data (e.g., text files) and produces a corpus as output. Each document in the corpus

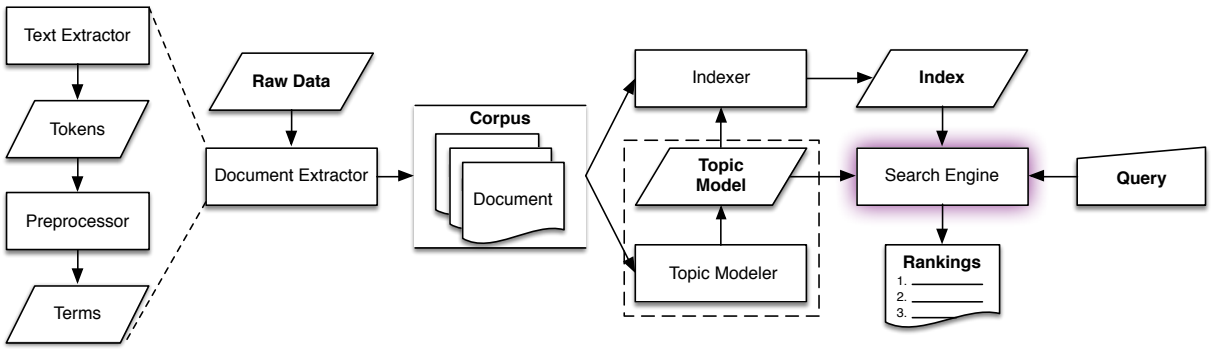


Figure 2.1: The general text retrieval process

contains the words associated to the origin (e.g., a file). The text extractor is the first part of the document extractor. It produces a token stream for each document in the data.

The preprocessor is the second part of the document extractor. It applies a series of transformations to each token and produces one or more terms from the token. The transformations commonly used are [Manning, Raghavan, and Schütze, 2008]:

1) **Split** separate tokens into constituent words by non-alphabetical characters or convention (e.g., "two-thirds" becomes "two" and "thirds")

2) **Normalize** replace each upper case letter with the corresponding lower case letter, or vice versa

3) **Filter** remove common words such as natural language articles (e.g., "an" or "the"), stop words, or short words

4) **Stem** remove prefixes and suffixes to leave just the root word (e.g., "name", "names", "named", and "naming" all reduce to "name" using the stemmer described by Porter [1980])

5) **Weigh** adjust the representation of a term in a document by some scheme, such as term-frequency inverse-document-frequency (tf-idf) [Salton and Buckley, 1988]

6) **Prune** remove terms that occur in, for example, over 80% or under 2% of the documents [Madsen, Sigurdsson, Hansen, and Larsen, 2004].

2.1.1.1 Document Extraction of Software

There are some additional considerations for applying text retrieval to software. In addition to the transformations already outlined, extended transformations [Marcus and Menzies, 2010; Marcus et al., 2004] commonly used in software are:

1) **Split** separate tokens into constituent words based on common coding style conventions (e.g., the use of camel case or underscores) and on the presence of non-letters (e.g., punctuation or digits)

2) **Filter** remove common words such programming language keywords, or standard library entity names

3) **Weigh** adjust the representation of a term in a document by some scheme, such as by the entity type [Bassett and Kraft, 2013].

2.1.2 Search Engine Construction and Document Retrieval

The right side of Figure 2.1 illustrates the retrieval process. The main component of the retrieval process is the search engine [Manning et al., 2008]. A search engine typically consists of an index and a classifier for ranking [Croft, Metzler, and Strohman, 2010]. Search engines based on topic models also need a trained model. The primary function of the search engine is to rank documents in relation to the query.

First, the engine transforms the corpus into an index. If the search engine relies on a topic model, then the engine uses the model to infer a document-topic distribution for each document.

Otherwise, the document-terms may have further transformations applied or be used directly as the index.

Next, the engine takes a pairwise classification of the query to each document in the index and ranks the documents according to similarity. I use a similarity measure for probability distributions for the pairwise comparisons.

2.1.3 Similarity measures

A similarity measure in text retrieval is useful for comparing two documents. Typically, documents are represented as a bag-of-words, or a term vector. Hence, I can use any vector-based similarity measure. Further, I can use discrete probability distributions, such as a document-topic proportion, as a vector. In the following definitions, P and Q are two discrete probability distributions of the same length K unless noted otherwise.

2.1.3.1 Cosine similarity

Cosine similarity (CS) is a commonly seen similarity measure [Croft et al., 2010] and is easy to implement:

$$\mathrm{CS}(P,Q) = \frac{P \cdot Q}{\|P\|\|Q\|} \tag{2.1}$$

where P and Q are term vectors. Since cosine similarity does not operate on probability distributions, it is not a suitable measurement for some models.

2.1.3.2 *Hellinger distance*

Hellinger distance can be used to measure the similarity between probability distributions, making it a good measure to use with topic models since topic models return topic-probability distributions for documents of interest.

I define Hellinger distance (H) as:

$$\mathrm{H}(P,Q) = \frac{1}{\sqrt{2}} \sqrt{\sum_{i=1}^{K} (\sqrt{P_i} - \sqrt{Q_i})^2} \tag{2.2}$$

where P and Q are probability distributions.

2.1.3.3 *Kullback-Leibler divergence*

Kullback-Leibler (KL) diverence is used for constructing topic models, but is not a good measure for similarity as $\mathrm{KL}(P,Q) \neq \mathrm{KL}(Q,P)$.

$$\mathrm{KL}(P,Q) = \sum_{i=1}^{K} P_i \ln \frac{P_i}{Q_i} \tag{2.3}$$

2.1.3.4 *Jensen-Shannon divergence*

Jensen-Shannon (JS) divergence measure addresses the KL-divergence issue by averaging the two KL measures together:

$$\mathrm{JS}(P,Q) = \frac{1}{2}\mathrm{KL}(P,M) + \frac{1}{2}\mathrm{KL}(Q,M) \tag{2.4}$$

where $M = \frac{1}{2}(P+Q)$. This makes JS-divergence an appropriate more measure for probability distributions over KL-divergence.

2.1.4 Evaluation measures

In the following section, I describe evaluation measures. I divide the measures into two groups: set-based measures and ranked-based measures.

For the following discussion, I define the following sets: $A = \{\text{relevant documents}\}$, the documents related to some query $q \in Q$; and $B = \{\text{retrieved documents}\}$, the documents retrieved for some query $q \in Q$.

2.1.4.1 Set-based measures

Set-based measures are useful when the text retrieval technique returns only a sub-set of the documents being searched over, i.e., the top 100 documents.

Recall

Recall is the fraction of relevant retrieved documents. I define recall as:

$$\text{recall} = \frac{|A \cap B|}{|A|} = P(B|A) \tag{2.5}$$

Precision

Precision is the fraction of retrieved documents that are relevant. I define precision as:

$$\text{precision} = \frac{|A \cap B|}{|B|} = P(A|B) \tag{2.6}$$

F-measure

In some situations, I may want to make trade-offs between precision and recall. For this, I can use the *F-measure*. The F-measure is a weighted harmonic mean of precision and recall:

$$\mathrm{F} = \frac{1}{\alpha \frac{1}{\text{precision}} + (1-\alpha)\frac{1}{\text{recall}}} = \frac{(\beta^2+1)\cdot \text{precision}\cdot \text{recall}}{\beta^2 \cdot \text{precision} + \text{recall}} \quad \text{where} \quad \beta^2 = \frac{1-\alpha}{\alpha} \tag{2.7}$$

where $\alpha \in [0,1]$ and thus $\beta^2 \in [0,\infty]$. A *balanced F-measure* equally weights precision and recall by $\alpha = 0.5$. When using a balanced F-measure, the formula simplifies to:

$$\mathrm{F} = \frac{2\cdot \text{precision}\cdot \text{recall}}{(\text{precision} + \text{recall})} \tag{2.8}$$

Precision at k

Precision at k is the calculated precision for the first k retrieved documents, excluding documents after the first k documents from the metric calculation. However, it has the disadvantage of not distinguishing between the rankings of the relevant document and is a count-based score.

$$\text{precision}(k) = \frac{|A \cap B_{1..k}|}{|B_{1..k}|} = P(A|B_{1..k}) \tag{2.9}$$

where $B_{1..k}$ is the top k documents retrieved. This is also written as *precision@k*.

2.1.4.2 Ranked-based measures

Since text retrieval techniques can also return a ranking of all documents searched (i.e., indexed), it is not beneficial to use set-based measures by only looking at the top-k documents, as in *precision@k*.

Average Precision (AP)

The average precision measure helps consider the order of returned documents by including the rank of relevant documents in its calculation. This measure is useful for scoring single queries.

$$\text{AP} = \frac{\sum_{i}^{|B|}(\text{precision(i)} \times \text{rel}(i))}{|A|} \tag{2.10}$$

where $rel(i)$ is the binary relevance function of the document at rank i, such that it is 1 if document at rank $i \in A$, and 0 otherwise.

Mean Average Precision (MAP)

Mean average precision uses AP to compute the scores over all queries, and is useful for summarizing the effectiveness of multiple queries [Manning et al., 2008].

$$\text{MAP} = \frac{1}{|Q|}\sum_{q}^{Q} AP(q) \tag{2.11}$$

where $AP(q)$ is the average precision for query q.

Mean Reciprocal Rank (MRR)

Reciprocal rank is often useful when there are few documents, or only one, relevant to the query. *Mean reciprocal rank* is an average of reciprocal ranks over different queries, hence it is useful for evaluating the effectiveness of a search engine [Croft et al., 2010].

$$\text{MRR} = \frac{1}{|Q|}\sum_{q}^{Q}\frac{1}{\text{bestRank}(q)} \tag{2.12}$$

where *bestRank* is a function equaling the rank of the first relevant item $a \in A$ for query q.

Discounted Cumulative Gain (DCG)

Discounted Cumulative Gain is a measure based on the assumptions that highly relevant documents are more useful than marginally relevant documents and that relevant documents with bad rank is not useful to the user [Järvelin and Kekäläinen, 2002]. Essentially, it penalizes, or discounts, when relevant documents are not highly ranked.

$$\text{DCG}(k) = rel(1) + \sum_{i=2}^{k}\frac{rel(i)}{log_2 i} \tag{2.13}$$

where $rel(i)$ is the graded relevance level of the document at rank i. A simple graded relevance function could be a binary judgement, such that it is 1 if document at rank $i \in A$, and 0 otherwise. Note that DCG does not penalize the relevance of the first retrieved document.

2.1.5 Statistical significance tests

The simplest experiment for text retrieval involves comparing two approaches: a baseline approach and an approach of interest. This design could use a set of common queries for each approach, obtaining matched pairs of *effectiveness measures*. A popular effectiveness measure

used in software is described by Poshyvanyk et al. [2007], and uses the rank of the first relevant document found.

I can form a null and alternative hypotheses for a one-tailed test using the evaluation measures discussed above to determine which of the approaches is better and by using a significance test on the effectiveness measures to determine if I should reject or accept a hypothesis. I can also use a two-sided test to determine if there is a difference between the two approaches.

There are several applicable ranked-based significance tests I can use. According to Croft et al. [2010], the most common ones are the sign test, the t-test, and the Wilcoxon sign-ranked test.

2.1.5.1 t-test

The t-test assumes a normal distribution of samples. That is, in the case of matched pairs, that the difference between effectiveness measures for each query is taken from a normal distribution. However, the t-test assumes that the effectiveness measure is interval, while effectiveness measures are typically ordinal, making the t-test an questionable choice depending on the effectiveness measure used.

2.1.5.2 Mann-Whitney U test

The Mann-Whitney U test is a non-parametric test similar to the t-test, but unlike the t-test, it does not require the assumption of normal distributions. This makes the Mann-Whitney U test a good choice when the effectiveness measure is ordinal and each sample is independent between two treatments.

2.1.5.3 *Wilcoxon signed-rank test*

The Wilcoxon sign-ranked test is non-parametric and does not make the same assumptions that the t-test does, making it a more desirable choice of a test when the effectiveness measure is ordinal and each sample is dependent between two treatments. I define the test as:

$$w = \sum_{i=1}^{N} sign(x_i - y_i) \times R_i \tag{2.14}$$

where N is the number of differences $\neq 0$, x and y are pairs of effectiveness measures, $sign(v)$ returns the sign of value v, R_i is the rank of that value. R_i is not the raw "rank" from the data; rather, it is the rank of the pair in a sorted list of their differences.

Effect size

An effect size can be derived from w, the Wilcoxon signed-rank test statistic [Kerby, 2014]. Given w, a rank correlation r can be defined as $r = w/S$, where S is the sum of all ranks.

2.1.5.4 *Friedman test*

Like the Wilcoxon sign-ranked test, the Friedman test is non-parametric and is desirable choice of a test when the effectiveness measure is ordinal. However, the Friedman test is useful when each sample has three or more dependent treatments.

2.1.6 Text Retrieval Models

In this section, I review commonly used text retrieval models. First, I review the boolean and vector space models. Then, I delve into the topic models that make up the basis of this work.

2.1.6.1 Boolean Model

The Boolean model is the simplest of the models used for constructing a search engine. This approach builds an index of the corpus by treating each document as a set of unique terms. Essentially, a boolean model weights all terms equally: either the term is in the document or it is not. A user constructs queries with single keywords joined by boolean expressions such as `AND`, `OR`, and `NOT`.

2.1.6.2 Vector Space Model

The Vector Space Model (VSM) is an algebraic model introduced by Salton, Wong, and Yang [1975]. VSM uses the $M \times N$ term-document matrix C directly as an index, where M is the number of unique terms in the corpus and N is the number of documents in the corpus. VSM represents each document in C as a vector of term weights, assigning words that appear in a document a weight by some weighting scheme and words that do not appear a weight of zero. That is, C_{ij} is the weight of the ith term in the jth document in the corpus C.

To search in the VSM, I transform a query document q (i.e., any document of interest) into a vector of term weights. Then, I perform pairwise comparisons of this document to each document in the index. I can use any vector-based measurement metric, such as cosine similarity, during the pairwise comparisons to measure the query document similarity. Documents in the index are then ranked according to how similar they are to the query document.

2.1.6.3 Topic Models

A topic model is a statistical model for discovering the abstract *topics* that occur in a corpus. For example, documents on Babe Ruth and baseball should end up in the same topic, while Dennis

Rodman and basketball should end up in another. Additionally, documents may also express multiple topics. That is, a document on Dennis Rodman could relate to multiple topics: basketball, tattoos, and vibrant hair coloring. In this section, I will describe common topic modeling algorithms and give a brief overview of the related works.

Latent Semantic Indexing

Latent semantic indexing (LSI) [Deerwester et al., 1990] is an indexing and retrieval methodology that extends the VSM. LSI relies on a mathematical technique called singular value decomposition (SVD) to find latent structure in a corpus represented in the VSM.

LSI begins with the $M \times N$ term-document matrix C, as in VSM. SVD computes C into three matrices by its rank r ($\leq min(M,N)$):

1) T (also called ϕ), an $M \times r$ term-topic vector matrix;
2) S, an $r \times r$ singular values matrix;
3) D (also called θ), an $N \times r$ document-topic vector matrix.

That is, $C = TSD^T$.

However, SVD allows for a reduction strategy to use smaller matrices that approximate C to reduce noise [Salton and McGill, 1983]. That is, SVD reduces the features in S by only keeping the first K largest values, where $K < r$, and removing the remaining values. Corresponding columns in T and rows in D of values removed from S are also removed. The result of this operation is a topic space approximation C_K, or $C \approx C_K = T_K S_K D_K^T$. Now, the dot product between two columns in C_K reflects the extent to which two documents (i.e., the columns) contain similar topics.

To search in LSI, I transform a query document q into the LSI topic space. First, I vectorize q into a vector of term weights, as in VSM. Next, because $C = TSD^T$, and hence $D = C^T S^{-1}$, I multiply q by TS^{-1} to transform q into a topic-document vector. Afterwards, I use this vector to make pairwise comparisons against all documents of C_K as before.

Extensions to SVD enable the algorithm to be *online* [Brand, 2006; Gorrell and Webb, 2005; Levey and Lindenbaum, 2000; Zha and Simon, 1999], thereby allowing for an online LSI. Online LSI allows for incremental updates to the model without needing to know about the documents prior to model construction. Řehůřek [2011] further extends the work of Brand [2006] to an LSI implementation that is both online and distributed. Halko, Martinsson, and Tropp [2011] outline a distributed algorithm, but it is not online.

Latent Dirichlet Allocation

Latent Dirichlet allocation (LDA) [Blei et al., 2003] is a fully generative model, assuming documents are generated according to a latent document-topic distribution and topic-word distribution. Of course, the goal of LDA is not to generate new documents from these distributions, although you certainly could, but is instead to infer the distributions of observed and unobserved documents. That is, LDA 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.

LDA assumes the following generative process:

1) Choose $\theta \sim \text{Dir}(\alpha)$.
2) Choose $\phi \sim \text{Dir}(\beta)$.

3) For each of the N words w_n:

   a) Choose a topic $z_k \sim \text{Multinomial}(\theta_d)$.

   b) Choose a word $w_n \sim \text{Multinomial}(\phi_z)$.

Here, α is the Dirichlet hyperparameter for the per-document topic distributions, β is the Dirichlet hyperparameter for the per-term topic distributions, θ is a $N \times K$ topic-document distribution matrix, with θ_d as the topic-document distribution for document d, and ϕ is a $M \times K$ term-topic distribution matrix, with ϕ_z as the term-topic distribution for topic z.

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 causes each topic to become 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.

To search in LDA, I transform a query document q into a topic probability distribution, similar to LSI. First, I vectorize q into a vector of term weights, as in VSM. Next, I *infer* from the model the topic probability distribution for the query. Afterwards, I use this distribution to make pairwise comparisons against all documents of θ.

Hoffman et al. [2010] introduce a version of LDA which is online. Zhai and Boyd-Graber [2013] introduce an extension of LDA in which the model does not need to know about the corpus

vocabulary prior to training. The Hierarchical Dirichlet Process (HDP) [Teh, Jordan, Beal, and Blei, 2006] is a similar model, in that it does not not need to have a pre-determined number of topics set.

2.2 Feature Location

Feature location is the act of identifying the source code entity or entities that implement a feature [Rajlich and Wilde, 2002]. Bug localization is the process of identifying source code entities that implement a bug, or an *unwanted* feature [Lukins et al., 2010]. Feature location is a frequent and fundamental activity for a developer tasked with changing a software system. Whether a change task involves adding, modifying, or removing a feature, a developer cannot complete the task without first locating the source code that implements the feature or where the feature is to be implemented and used. Often, an approach for feature location technique (FLT) interchangeable for a bug localization technique (BLT).

The most closely related work to this work is Rao et al. [2013]. Rao et al. [2013] also target the problem of building topic models, introducing an incremental framework for bug localization. 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 [2013] notes that these algorithm modifications have limitations and thus models may need to be periodically retrained.

Dit et al. [2013] provide a taxonomy and survey of feature location in source code covering the scope of FLTs. They identify 89 works related to feature location in their systematic literature survey and extract 7 dimensions for their taxonomy. I use the primary dimension, type of analysis, for categorization purposes. The different types of analysis are: dynamic, static, historical (mining), and textual.

Dynamic FLTs use information from a system's execution, such as stack traces [Moreno, Treadway, Marcus, and Shen, 2014]. Static FLTs instead use semantic information extracted directly from source code, such as method call graphs [Saul, Filkov, Devanbu, and Bird, 2007]. Historical FLTs use software repository mining to extract meaningful information, such as in Cubranic, Murphy, Singer, and Booth [2005]. Textual FLTs use textual information extracted from software artifacts, such as source code comments, identifiers, and literals [Biggers et al., 2014]. Textual FLTs are the most closely related category of my work and comprise the remainder of this section.

Currently, developers rely on tools such as `grep` to find relevant source code entities. Petrenko, Rajlich, and Vanciu [2008] develop a `grep`-based FLT. However, Ko et al. [2006] show that developers fail using this type of searching upwards to 88% of the time. Text retrieval techniques, such as topic modeling, show promise in remedying this problem [Marcus et al., 2004].

Marcus et al. [2004] use an FLT based on Latent Semantic Indexing (LSI) [Deerwester et al., 1990] to find concepts based on queries from the user, and modules within the system. They found that concepts in the code were identifiable with user specified terms and identifiers as well as an easier build process. LSI-based FLTs have been widely used by others [Cubranic et al., 2005; Eaddy, Aho, Antoniol, and Guéhéneuc, 2008; Liu, Marcus, Poshyvanyk, and Rajlich, 2007; Poshyvanyk, Gueheneuc, Marcus, Antoniol, and Rajlich, 2006; Poshyvanyk and Marcus, 2007; Scanniello and Marcus, 2011].

Lukins et al. [2008] introduce an FLT based on latent Dirichlet allocation (LDA) [Blei et al., 2003] and find that it outperforms the LSI-based FLT by Poshyvanyk et al. [2007]. They use the LDA inference technique to infer the topic distributions of queries, i.e., bug reports. Later, they show LDA to be appropriate for software systems of any size [Lukins et al., 2010].

Biggers et al. [2014] investigate 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 show that configuration parameters taken from the machine learning and natural language processing (NLP) communities are not good choices for software. Dit, Guerrouj, Poshyvanyk, and Antoniol [2011] show the need for better term splitting techniques for software TR.

Bassett and Kraft [2013] explore new term weighting schemes based on the structural information available in source code. Namely, they find 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) [Salton and Buckley, 1988]. Saha, Lease, Khurshid, and Perry [2013] show that using structural information provides improvement over tf-idf, as well. Saha, Lawall, Khurshid, and Perry [2014] extend their work to show that improvements using structural information apply to both Java and C. Zhou, Tong, Chen, and Han [2017] show that a parts-of-speech weighting method, with a particular focus on the nouns, increases the accuracy of a BLT.

Eddy, Kraft, and Gray [2017] extend Bassett and Kraft [2013] to include new weighting schemes, which consider a method's documentation comments, parameter names, implementation comments (i.e., inline comments), and the variables within the method, as well as a class' name, documentation comments, and field names within the class. Their findings conclude that the context of a method (i.e., the containing class) may be beneficial to a method-level FLT.

Combining textual and static techniques shows improvement over using one or the other alone. Shao, Atkison, Kraft, and Smith [2012] combine LSI with call graph information and find that

the call graph information increases the accuracy over plain LSI. Likewise, Ali, Sabane, Gueheneuc, and Antoniol [2012] use binary class relationships in combination with LSI and VSM to further improve their FLT.

More recent work has focused on integrating multiple sources of information. Dit, Revelle, and Poshyvanyk [2012] combine textual, dynamic, and a new category, *mining*, to increase the effectiveness of FLTs. Wang, Khomh, and Zou [2013] utilize stack traces by using a Bayesian networks to adjust the rank of a file based on three features that determine the probability of the file being buggy. Moreno et al. [2014] also use stack traces with a VSM-based FLT to improve their accuracy of their FLT. Youm, Ahn, and Lee [2017] combine changesets, stack traces, bug reports, and source code for their VSM-based BLT.

Sisman and Kak [2013] present work on an automatic query reformulation approach for changing a user's query to achieve better ranking in their BLT. Chaparro, Florez, and Marcus [2017] performed manual coding of queries in order to generate reduced queries that improve the accuracy of various baseline TR-based approaches [Just, Jalali, and Ernst, 2014, Moreno et al. [2014], Wong, Xiong, Zhang, Hao, Zhang, and Mei [2014], Mills, Bavota, Haiduc, Oliveto, Marcus, and Lucia [2017]] on average of 147%.

2.3 Developer Identification

In this section I review literature on developer identification. As noted by Shokripour et al. [2013], there are two broad categories of work in this area: activity-based approaches and location-based approaches. An activity-based approach uses information gained from a developers *activity*, e.g., the change requests they have worked on in the past. Location-based approaches resemble a feature location technique in that they rely on source code entity information to derive a

developer, e.g., which developer has worked on the related classes in the past? There are multiple ways to determine the ownership of a source code entity [Bird et al., 2011; Corley et al., 2012; Hossen et al., 2014; Kagdi et al., 2012]. I refer the reader to Section 2.1 for a background on how a typical feature location techniques works.

2.3.1 Activity-based approaches

Mockus and Herbsleb [2002] present Expertise Browser to locate expertise. The browser uses units of experience called Experience Atoms (EA) extracted from code changes. The number EAs in a certain domain or file determines the developer's expertise on that domain or file.

Cubranic and Murphy [2004] propose a machine learning approach that uses text categorization on change request descriptions. Cubranic and Murphy [2004] also report on heuristics used for classification of change requests. Anvik et al. [2006] use machine learning in an approach for semi-automated triage by using change request history to learn which requests a developer completes. Liu, Tian, Yu, Yang, Jia, Ma, and Xu [2016] use an LDA-based approach to find bug reports that are similar to one another and take into consideration how long each similar bug took to fix before making recommendations.

Anvik and Murphy [2007] conduct an empirical evaluation of two approaches for recommending: one that uses software repository mining, and one that uses change request repository mining. The evaluation finds that the software repository approach has higher precision, but lower recall than the change request repository approach.

Canfora and Cerulo [2006] propose an information retrieval-based approach that indexes the textual description of previously resolved change requests. The documents represent the developer's descriptions of completed change requests. Matter et al. [2009] take a slightly different approach

and extract developer documents from source code history. The history-based document measures how active a developer is with a set of words in a VSM.

Linstead, Rigor, Bajracharya, Lopes, and Baldi [2007] report on the use of Author-Topic modeling [Steyvers, Smyth, Rosen-Zvi, and Griffiths, 2004]. The Author-Topic model augments existing topic modeling [Blei et al., 2003] to model the distribution of authors over topics in addition to topics over documents. Linstead et al. [2007] use bug reports to attribute authorship to developers. The topics allow for comparison of developers based on their contributions to a topic.

Guo, Zimmermann, Nagappan, and Murphy [2011] report on a large-scale analysis of bug reassignment in Microsoft Windows Vista operating system project. The study finds five primary reasons for reassignment: finding the root cause, expertise identification, low quality reports, difficulty in determining a proper fix, and workload balance. These reasons suggest considerations in triage that can potentially improve automatic assignment. The study also validates previous observations [Guo, Zimmermann, Nagappan, and Murphy, 2010] that reassignment is not always harmful, but can be beneficial in finding the best developer to complete a request.

Somasundaram and Murphy [2012] propose an approach combining LDA with a machine learning algorithm for automated change request categorization. Improving categorization of change requests shows potential benefits to triaging change requests by reducing the space of expertise that requires consideration. Knowing which component a request belongs to provides two benefits: knowing the component reduces the time-to-fix of a report [Guo et al., 2011], and only members of the team associated with the component need consideration for recommendation. The paper reports a comparative study on three variations of categorization approaches and finds LDA improves categorization over other approaches [Anvik et al., 2006].

Jeong et al. [2009] use a Markov chain-based learning algorithm that considers bug reassignment information. Using their bug reassignment model, they reduce the possibility of a bug reassignment by ensuring the bug is assigned to the correct developer the first time. They also show that a bug reassignment increases the time until completion by about 100 days. Bhattacharya, Neamtiu, and Shelton [2012] further employ this idea using different learning algorithms incrementally improves triaging bugs the first time.

2.3.2 Location-based approaches

McDonald and Ackerman [2000] present a heuristic-based recommender system named Expertise Recommender. The recommender uses heuristics derived in a previous industrial study [McDonald and Ackerman, 1998] on how developers locate expertise. 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.

Fritz, Murphy, and Hill [2007] investigate whether a programmer's activity indicates knowledge of code in an empirical study on nineteen professional Java programmers. They study finds that the frequency and recency of interaction is an indicator of the expertise a developer has on portions of code. They also report on interviews with developers, finding other indicators that may improve expertise models based on source code interaction. The indicators include authorship, role of source code, and the programmer's task.

Minto and Murphy [2007] propose an approach implemented in a tool named Emergent Expertise Locator. The approach uses the matrices to produce requirements coordination by Cataldo, Wagstrom, Herbsleb, and Carley [2006]. The matrices represent 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. They compare the results to the approach by McDonald and Ackerman [2000], and find higher precision and recall in their own approach.

Kagdi, Hammad, and Maletic [2008] 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. [2011] finds that measuring ownership in this way correlates low ownership with post-release defects.

Rahman and Devanbu [2011] use the provenance features of Git to track the ownership of individual lines of source code. They then study the impacts of ownership and experience on software quality by comparing the ownership and experience characteristics of "implicated code" (lines of code changed to fix bugs) to those of "normal code." This paper reports an association between strong ownership by a single developer and implicated code, and an association between lack of specialized experience on a particular file and implicated code in that file. This suggests that the best suited developer for change requests is the developer with the most ownership (i.e., expertise).

Ma, Schuler, Zimmermann, and Sillito [2009] evaluate the proposed approach by Schuler and Zimmermann [2008]. The paper proposes a approach of six heuristics: two based on implementation expertise and four based on usage expertise. The results show usage-expertise-based recommendations have an accuracy comparable to implementation-based recommendations.

Linares-Vásquez, Hossen, Dang, Kagdi, Gethers, and Poshyvanyk [2012] present an approach that does not require mining the software history nor learning from previously completed change requests. Using the author indicated in source code comments with an LSI-based FLT, they are able to identify a correct developer. Hossen et al. [2014] extend this approach to also include change proneness to adjust the rank of relevant source code entities before selecting a developer.

Weissgerber, Pohl, and Burch [2007] present three visualization techniques that can help a triager identify the developer most appropriate for a task. Bortis and Van der Hoek [2013] present an approach that tags bugs to help developers explore relevant bugs. Tamrawi, Nguyen, Al-Kofahi, and Nguyen [2011] present an incremental DIT approach based on fuzzy sets. Like Bassett and Kraft [2013], Shokripour et al. [2013] show that using a term weighting scheme increases the accuracy of a DIT.

Ouni, Kula, and Inoue [2016] present a DIT for use in finding a code reviewer. Their approach uses the source code change history to build an expertise model of each developer's previously changed files and the files changed in previously completed code reviews. Their evaluation takes into consideration only the changes or reviews completed prior to the review of interest.

3. METHODOLOGY

In this chapter, I describe my approach and the three studies that address the respective research problems. I first describe the changeset-based approach and compare it to the approach of a snapshot-based topic model search engine. I then give my reasoning for why changesets may be good choice for a training corpus. Then, I discuss the datasets and benchmarks used throughout this work. Next, I describe my approach for an application of topic models for feature location and how it contrasts to the state-of-the-practice. I then discuss work on the application of topic models for developer identification. Finally, I discuss an approach for using a single topic model for both of these tasks.

3.1 Modeling Changeset Topics

In this section I describe my approach to a modeling changeset topics and how I apply them for feature location and developer identification.

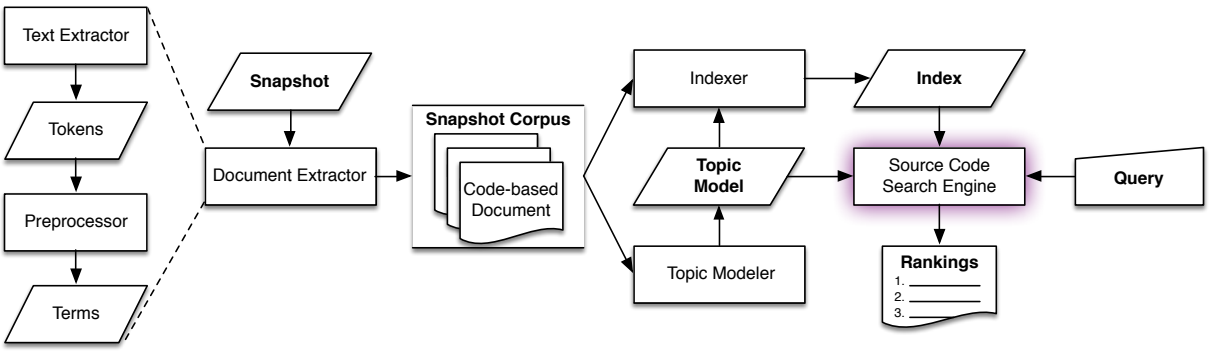


Figure 3.1: Constructing a search engine with snapshots

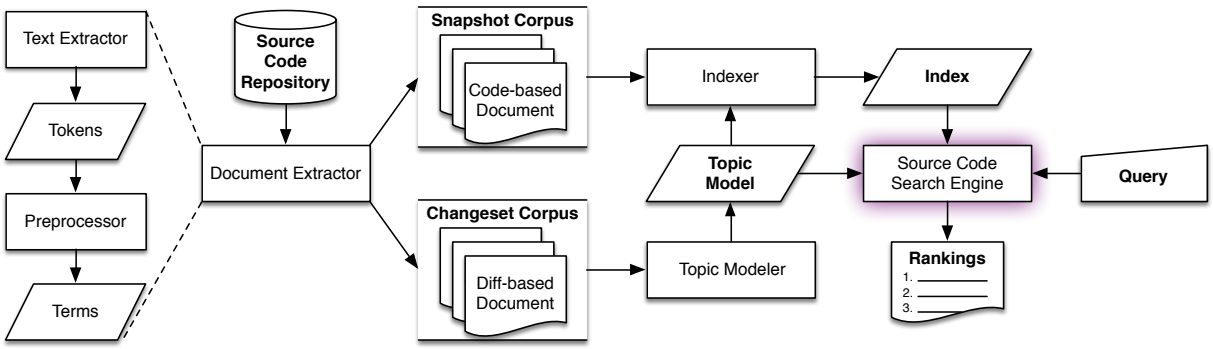


Figure 3.2: Constructing a search engine from changesets

3.1.1 Approach overview

The overall difference in my approach and the standard approach described in Section 2.1 is minimal. For example, compare Figures 3.1 and 3.2. In the changeset approach, I only need to replace the training documents while the remainder of the approach remains the same.

The changeset-based approach requires two types of document extraction: a corpus at a point of interest, or snapshot, and every changeset in the source code history leading up to the same point of interest. The left side of Figure 3.2 illustrates the dual-document extraction approach.

The document extraction process for the snapshot remains the same as covered in Section 2.1, but may be modified to fit the task required. The document extractor for the changesets parses each changeset for the removed, added, and context lines. From there, the text extractor tokenizes each line. The same preprocessor transformations occur for both snapshot and changeset corpora. This helps ensure that the snapshot vocabulary is a subset of the changeset vocabulary [Corley, Kashuda, May, and Kraft, 2014].

The right side of Figure 3.2 illustrates the retrieval process. The key intuition to my approach is that a topic model such as LDA or LSI can infer *any* document's topic proportions regardless

of the documents used to train the model. This is also what determining the topic proportions of a user-created query has relied on in traditional TM-based FLTs or DITs. Likewise, so are other unseen documents. In my approach, the seen documents are changesets and the unseen documents are the source code entities of the snapshot.

I train a topic model on the changeset corpus and use the model to index the snapshot corpus. Note that I never construct an index of the changeset documents used to train the model. I only use the changesets to continuously update the topic model and only use the snapshot for indexing.

To leverage the online functionality of the topic models, I can also intermix the model training, indexing, and retrieval steps. First, I initialize a model in online mode. I update the model with new changesets whenever a developer makes a new commit. That is, with changesets, I incrementally update a model and can query it at any moment. This allows for a *historical simulation* of how a changeset-based approach would perform in a more realistic scenario than batch training allows.

3.1.1.1 Feature Location with Changeset Topics

Application of the constructed changeset-based topic model for feature location does not require any more work than described above. The snapshot I will index is a release version, and I have two options for this: the release source code package or the state of the source code repository at the commit tagged with the corresponding release identifier. I choose the latter option to ensure that the vocabulary of the indexed corpus is a subset of the modeled corpus [Corley et al., 2014], although the former option is entirely possible.

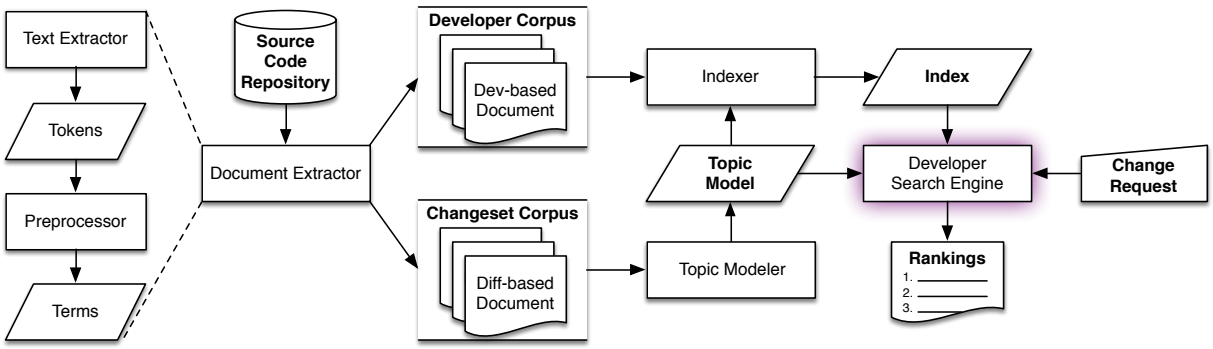


Figure 3.3: Developer identification using changesets

3.1.1.2 Developer Identification with Changeset Topics

Application of the constructed changeset-based topic model for developer identification varies slightly from the general approach, as seen in Figure 3.3. Following Matter et al. [2009], each developer has their own document, or profile, consisting of all changesets they have committed to the source code repository. That is, the snapshot in this case is a corpus of developer documents that consists of all lines a particular developer has changed. As with the FLT approach, this ensures that each developer profile indexed is a subset of the modeled corpus.

3.1.1.3 Combining and Configuring Changeset-based Topic Models

The changeset topic modeling approach requires three types of document extraction: one for the snapshot of the state of source code at a commit of interest, such as a tagged release; one for every changeset in the source code history leading up to that commit; and a developer profile of all lines each individual developer changed in their changesets. The left side of Figure 3.4 illustrates the tri-document extraction approach.

The document extraction process for snapshot and changeset corpora remain the same as

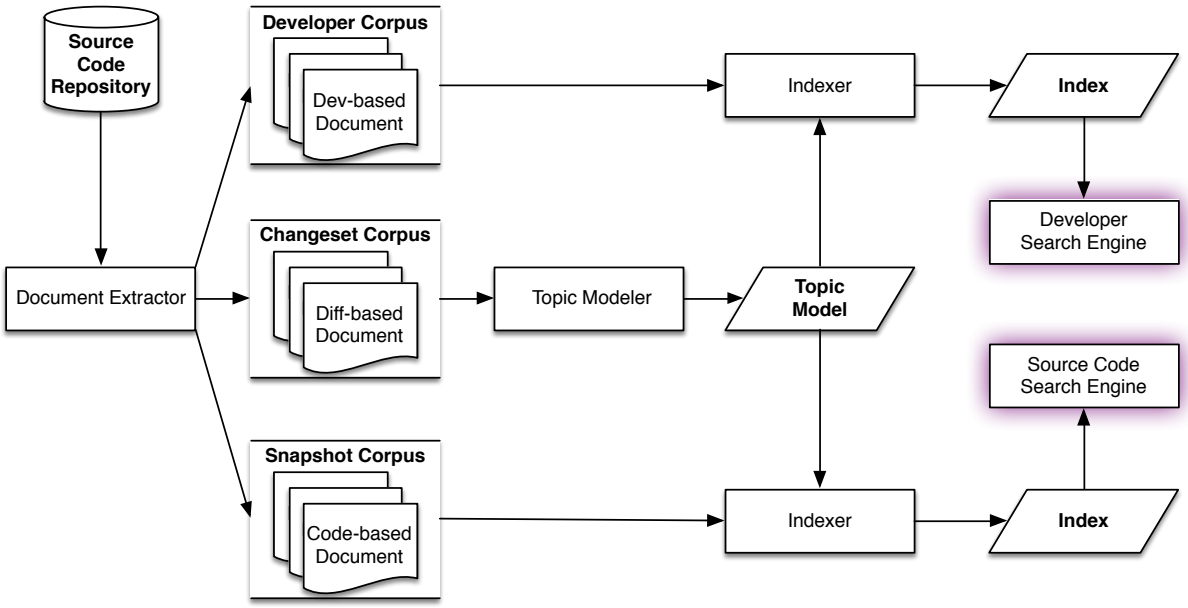


Figure 3.4: Combining changeset-based feature location and developer identification

covered in Section 3.1.1.1. The document extraction process for the corpus of developer profiles remains the same as covered in Section 3.1.1.2.

The right side of Figure 3.4 illustrates the retrieval process. For brevity, the queries and ranking do not appear in the diagram, as they remain the same as described for each search engine in their respective sections, Sections 3.1.1.1 and 3.1.1.2. I train a topic model on the changeset corpus and construct search engines for each task separately. In the source code search engine I build an index from the snapshot corpus. In the developer search engine I build an index from the developer corpus.

3.1.2 Why changesets?

I choose to train the model on changesets, rather than another source of information, because they represent what I are primarily interested in: program features. A single changeset gives me a

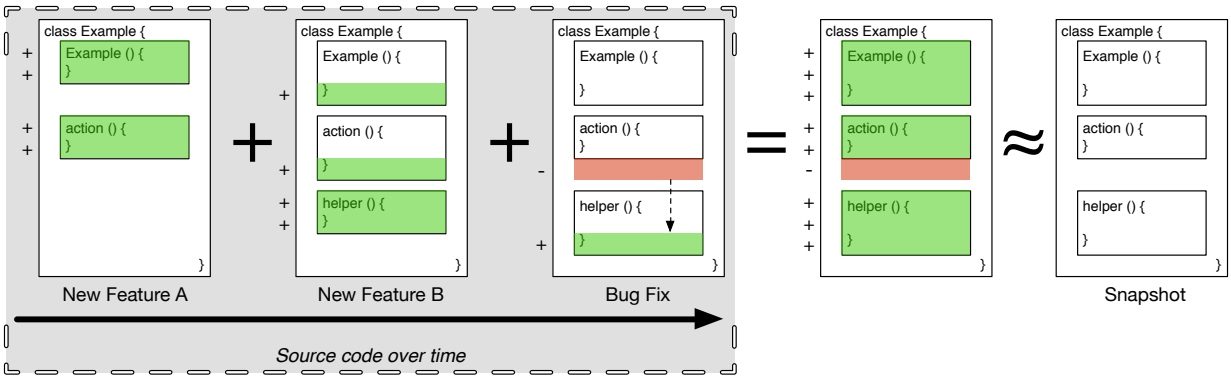


Figure 3.5: Changesets over time approximate a Snapshot. Green lines (denoted on the left with +) denote addition lines, and red lines (denoted on the left with −) denote removals lines, while uncolored lines (not denoted) represent context lines.

view of an addition, removal, or modification of a single feature. A developer may, to some degree, 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 I consider every changeset affecting a particular source code entity, then I gain a sliding-window view of that source code entity over time and the contexts in which those changes took place. Figure 3.5 shows an example, where green areas denote text added and red areas denote text removed in that changeset. Here, the summation of all changes affecting a class over its lifetime would approximate the same words in its current version.

Changeset topic modeling is akin to summarizing code snippets with machine learning [Ying and Robillard, 2013], where in my case a changeset gives a snippet-like view of the code required to complete a task. Additionally, Vasa, Schneider, and Nierstrasz [2007] observe that code rarely changes as software evolves. The implication is that the topic modeler will see changesets containing the same source code entity only a few times – perhaps only once. I note that the initial commit of a file is the entire file at the time, conceptually much like a snapshot. Since topic modeling a snapshot only sees an entity once, topic modeling a changeset can miss no information.

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. [2007] also observe that code is less likely to be removed than it is to be changed. This implies that the noisy information would likely remain in both snapshot-based models and changeset-based models.

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

3.2 Study Design

In this work, I introduce topic-modeling-based FLT and DIT in which I incrementally build the model from source code changesets. By training an online learning algorithm using changesets, my FLT and DIT maintain up-to-date models without incurring the non-trivial computational cost associated with retraining a snapshot-based topic model from scratch. In this section, I describe the design of my study in which I compare my new methodology with the current practice.

3.2.1 Definition and Context

The primary goal of this work is to evaluate the performance and reliability of topic models built on the source code histories, i.e., changeset-based topic modeling, in order to move towards more robust approaches that can be utilized by practitioners. For this work, I pose the following research questions. First, with respect to *Research Problem 1* (RP1), applying my approach to feature location, I ask:

RQ 1.1 Is a changeset-based FLT as accurate as a snapshot-based FLT?

RQ 1.2 Does the accuracy of a changeset-based FLT fluctuate as a project evolves?

Then, for *Research Problem 2* (RP2), applying my approach to developer identification, I ask:

RQ 2.1 Is a changeset-based DIT as accurate as a snapshot-based DIT?

RQ 2.2 Does the accuracy of a changeset-based DIT fluctuate as a project evolves?

Finally, I determine for *Research Problem 3* (RP3) whether I can gain model re-use for both tasks by asking the following:

RQ 3.1 Can we use the same topic model in more than one context effectively?

RQ 3.2 What are the effects of using different portions of a changeset for corpus construction, such as added, removed, context lines, and the commit message?

3.2.2 Datasets and Benchmarks

For the first two Research Problems, there do exist various datasets and benchmarks for each [Dit, Holtzhauer, Poshyvanyk, and Kagdi, 2013; Kagdi et al., 2012; Linares-Vásquez et al., 2012; Moreno et al., 2014]. However, *Research Problem 3* (RP3) introduces a complication to using these benchmarks: requiring high overlap of the benchmarks. The overlap of these benchmarks is small or non-existent, making it difficult to determine whether a technique is performing well because of the approach or if it happens to just be a challenging query for that technique. Further, these datasets often only contain processed data, e.g., Moreno et al. [2014] does not provide original issue numbers of the extracted data. I have created my own benchmark fit for evaluating both an FLT and DIT.

Table 3.1: 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

The 6 subjects of the 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 3.1 summarizes the sizes of each system's corpora and dataset.

I chose these systems for this work because developers use descriptive commit messages that allow for easy traceability-linking to issue reports. These systems are also studied by other researchers [Moreno et al., 2014]. Further, all projects use JIRA as an issue tracker, which has been found to encourage more accurate traceability link recovery [Bissyande, Thung, Wang, Lo, Jiang, and Reveillere, 2013]. Each system varies in domain and in size, in terms of developers, changesets, and number of source code files.

To build this dataset I mine the Git repository for information about each commit: the

[1] http://zookeeper.apache.org/bookkeeper/
[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

committer, message, and files changed. I use the files changed information during the location-based approach. Using the message, I extract the traceability links to issues in JIRA with the case-insensitive 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, I extract the release version the issue marked as fixed in. I ignore issues that are not marked with a fixed version. I also extract the title and description of the issue, which I use as the query for that particular issue.

I 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. On occasion, an issue will be addressed multiple times by two developers, and I consider both developers as valid selections when ranking queries. I do not consider the case when the change was submitted by another developer as a patch, and then applied and committed by a core contributor. In this case, I assume that the core contributor, being a subject matter expert, understands and agrees with the change since they were the one that committed the changeset.

3.2.3 Evaluation

In this section, I discuss the evaluations for each research question.

3.2.3.1 Feature Location

To answer *RQ 1.1*, I evaluate two batch-trained models: one trained on a snapshot corpus, and the other on a changeset corpus.

For snapshots, the process is straightforward and corresponds to Figure 3.1. I first train a

model on the snapshot corpus using batch training. That is, the model can see all documents in the corpus at once. Then, I infer an index of topic distributions with the snapshot corpus. For each query in the dataset, I infer the query's topic distribution and rank each entity in the index with pairwise comparisons.

For changesets, the process varies slightly from a snapshot approach, as shown in Figure 3.2. First, I train a model of the changeset corpus using batch training. Second, I infer an index of topic distributions with the snapshot corpus. Note that I *do not* infer topic distributions with the changeset corpus. Finally, for each query in the dataset, I infer the query's topic distribution and rank each entity in the snapshot index with pairwise comparisons.

To answer *RQ 1.2*, for the historical simulation, I must take a different approach. I first determine which commits relate to each query (or issue) and partition mini-batches out of the changesets. I then proceed by initializing a model for online training with no initial corpus. I then update the model with each mini-batch. Then, I infer an index of topic distributions with the snapshot corpus at the commit the partition ends on. I also obtain a topic distribution for each query related to the commit. For each query, I infer the query's topic distribution and rank each entity in the snapshot index with pairwise comparisons. Finally, I continue by updating the model with the next mini-batch.

Since my dataset is extracted from the commit that implemented the change, my partitioning is inclusive of that commit. That is, I update the model with the linked commit and infer the snapshot index from that commit. This allows the evaluations to capture any entities added to address the issue report, as well as changed entities, but does not capture any entities removed by the change.

3.2.3.2 Developer Identification

To answer *RQ 2.1*, I take the approach described by Matter et al. [2009], but instead of using a VSM, I apply the approach using LDA. This evaluation requires a snapshot corpus, a changeset corpus, and a corpus of *developer profiles*, as described in Section 3.1.1.2. Using the developer profile approach rather than a heuristic-based approach, such as Bird et al. [2011], allows for me to construct a more fair evaluation of batch-trained models by only varying the training corpus, i.e., the snapshot corpus or the changeset corpus.

For developer identification using snapshots, I first build a topic model for searching over the source code snapshot (i.e., like the FLT evaluation). Then, I infer an index of topic distributions with a corpus of developer profiles. Note that I *do not* infer topic distributions with the snapshot corpus used to train the model. For each query in the dataset, I infer the query's topic distribution and rank each entity in the index with pairwise comparisons.

For changesets, the process varies slightly from a snapshot approach, as shown in Figure 3.3. First, I train a model of the changeset corpus using batch training. Second, I infer an index of topic distributions with the developer profiles. Note, again, that I *do not* infer topic distributions with the changeset corpus used to train the model. Finally, for each query in the dataset, I infer the query's topic distribution and rank each entity in the developer profiles index with pairwise comparisons.

To answer *RQ 2.2*, the historical simulation, the approach is analogous of the approach described in the previous section, Section 3.2.3.1. The approach for developer identification, however, needs to vary from the previous approach only with respect to indexing. With each mini-batch, instead of indexing a snapshot corpus of the commit at the end of the mini-batch, I index of topic distributions with the developer profiles at that commit. Note that each developer

Table 3.2: Case study factors

Factors	Values
K	{100, 200, 500}
α	$\{\frac{1}{K}, \frac{2}{K}, \frac{5}{K}$, learned$\}$
η	$\{\frac{1}{K}, \frac{2}{K}, \frac{5}{K}$, learned$\}$
Task	{FLT, DIT}

Table 3.3: Text sources

Text source	Include?
Added lines	{yes, no}
Removed lines	{yes, no}
Context lines	{yes, no}
Commit message	{yes, no}

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.

3.2.3.3 Combining and Configuring Changeset-based Topic Models

In this work, I reuse the already-created framework from the previous two research areas covered in this work. I will not need to instantiate snapshot models for this work. Where this work will differ from the previous two research areas is in how I construct the corpus and topic model.

For *RQ 3.1*, I want to find if the two approaches can rely on the same model with minimal interference from one another's requirements. For example, the FLT task may perform better with less topics, while the DIT task may require more topics for optimal performance. Table 3.2 outlines the factors about the model construction I will consider, giving me 48 possible combinations. The factors α and η vary between several common values, but also include automatic learning of these two hyper-parameters [Hoffman et al., 2010]. As in sections 3.2.3.1 and 3.2.3.2, the changeset corpus construction for *RQ 3.1* does not change and includes the entire `diff`, but does not include the message.

While *RQ 3.1* focuses on model construction, *RQ 3.2* focuses on training-corpus construction. For *RQ 3.2*, there are several combinations to consider. While thus far I have always included the

entire changeset when training, it may be beneficial to consider only including portions (i.e., added lines only) while excluding other portions (i.e., removed lines). It may also be beneficial to include the natural language text of the commit message. Table 3.3 outlines the 16 combinations over the 4 types of changeset text. One combination is invalid because it constructs an empty corpus and thus impossible to train a topic model, leaving 15 total combinations.

3.2.4 Setting

The document extraction process is shown on the left side of Figure 3.2. I implemented the document extractor in Python v2.7 using the Dulwich library[7] for interacting with the source code repository. I extract documents from both a snapshot of the repository at a tagged commit and each commit reachable from that tag's commit. The same preprocessing steps are employed on all extracted documents, regardless of corpus source.

To extract text from the changesets, I use the `git diff` between two commits. In the changeset text extractor, I extract all text related to the change, e.g., context, removed, and added lines; metadata, such as commit messages, are ignored unless stated otherwise. Note that I do not consider where the text originates from, only that it is text of the changeset.

After extracting tokens, I split the tokens based on camel case, underscores, and non-letters. I only keep the split tokens; original tokens are discarded to remove duplication of information and keep the corpus vocabulary small. I normalize to lower case before filtering non-letters, English stop words [Fox, 1992], Java keywords, and words shorter than three characters long. I do not stem words.

I implemented the modeling using the Python library Gensim [Řehůřek and Sojka, 2010],

[7] http://www.samba.org/~jelmer/dulwich/

version 0.12.1. I use the same configurations on each subject system. I do not try to adjust parameters between the different systems to attempt to find a better, or best, solution; rather, I leave them the same to reduce confounding variables. I do realize that this may lead to topic models that may not be best-suited for each task on a particular subject system. This constraint gives me confidence that the measurements collected are fair and that the results are not influenced by selective parameter tweaking. Again, the goal is to show the performance of the changeset-based topic model against snapshot-based topic model under the same conditions. I do, however, adjust parameters during configuration sweeps for RP3 and those are noted where appropriate.

Gensim's LDA implementation is based on an online LDA by Hoffman et al. [2010] and uses variational inference instead of a collapsed Gibbs sampler. Unlike Gibbs sampling, in order to ensure that the model converges for each document, I set the model to take 1000 iterations over a document during the inference step, and as many passes over the corpus as needed until there is little improvement in the evidence lower bound. That is, I allow the model to behave as a batch, or offline, model where possible. I set the following LDA parameters for all systems: 500 topics, a symmetric $\alpha = 1/K$, and a symmetric $\eta = 1/K$. These are default values for α and η in Gensim, and have been found to work well for the FLT task [Biggers et al., 2014]. Again, I adjust all three of these parameters for RP3, but these values are used in all research problems unless stated otherwise.

For historical simulation (RP1 and RP2), since I must use online training, I found it beneficial to consider two new parameters for online LDA: κ and τ_0. As noted in Hoffman et al. [2010], it is beneficial to adjust κ and τ_0 to higher values for smaller mini-batches, e.g., a single changeset. These two parameters control how much influence a new mini-batch has on the model when training. I follow the recommendations in Hoffman et al. [2010], choosing $\tau_0 = 1024$ and $\kappa = 0.9$ for all

systems, because the historical simulation often has mini-batch sizes in single digits. Additionally, since I are operating in a truly online mode, I cannot take multiple passes over the entire corpus as that would defeat the purpose of a historical simulation.

For all training and querying of the LDA models I hold two things constant: first, the pseudo-random number generator seed is set to 1 before each model is instantiated, and second, I ensure that corpus documents are always read in the same order for each pass over them. This corpus order is maintained on *all* documents used in this study, including training data and query data. The first, the random seed, is a result of the random and generative nature of LDA, though no known study yet exists showing its impact on model performance for software engineering tasks. The second requirement stems from the same observations first made by Agrawal, Fu, and Menzies [2018]: LDA suffers from "order effects" when the training data is shuffled.

3.2.5 Data Collection and Analysis

To evaluate the performance of a topic-modeling-based FLT and DIT I cannot use measures such as precision and recall. This is because each approach creates rankings pairwise, causing every entity to appear in the rankings. Poshyvanyk et al. [2007] define an effectiveness measure for topic-modeling-based FLTs, which is also applicable to DIT. This effectiveness measure is the rank of the first relevant document, whether that document represents a source code entity or developer profile, and represents the number of documents entities a developer or triager would have to view before reaching a relevant one. Most importantly, this effectiveness measure allows evaluating the approach by using various rank-based metrics and statistical methods, such as the mean reciprocal rank (MRR) and the Wilcoxon signed-rank test.

3.2.5.1 Feature Location

To answer *RQ 1.1*, I run the experiment on the snapshot and changeset corpora as outlined in Section 3.2.3.1. I then calculate the MRR between the two sets of effectiveness measures. I use the Wilcoxon signed-rank test with Holm correction to determine the statistical significance of the difference between the two rankings. To answer *RQ 1.2*, I run the historical simulation as outlined in Section 3.2.3.1 and compare it to the results of batch changesets from *RQ 1.1* using the same measures.

I must note that when performing evaluations on FLTs, it is possible to encounter a query that will fail to retrieve any related documents. This is related to both the approach and goldset extraction process. Since the approach indexes only on the snapshot corpus for a particular commit of interest, it is possible that a file changed to fix a particular issue in the goldset no longer exists or was never committed to source control by maintainers. I exclude these queries from all analysis. One alternative to exclusion would be to penalize the score by assigning maximum rank to the failed query. I choose exclusion over penalization as there is no evidence supporting penalization to the best of my knowledge.

3.2.5.2 Developer Identification

To answer *RQ 2.1*, I run the experiment on the snapshot and changeset corpora as outlined in Section 3.2.3.2. I then calculate the MRR between the two sets of effectiveness measures. I use the Wilcoxon signed-rank test with Holm correction to determine the statistical significance of the difference between the two rankings. To answer *RQ 2.2*, I run the historical simulation as outlined in Section 3.2.3.2 and compare it to the results of batch changesets from *RQ 2.1*. Again, I calculate the MRR and use the Wilcoxon signed-rank test.

3.2.5.3 Combining and Configuring Changeset-based Topic Models

For both research questions, I run the experiment of each task across all possible configurations from two sets of configurations, shown in Tables 3.2 and 3.3. The first set of configurations varies the model parameters, while the second set of configurations varies the corpus construction inclusion parameters. Each set is associated with a "default" configuration for the other, e.g., all variations of model configurations use the same corpus configuration and vice versa. The default configurations are the same used throughout this thesis.

To answer *RQ 3.1*, I collect the effectiveness measures of each configuration variation and use them calculate the MRR. For each task, I create a pair of effectiveness measures using the optimal configuration, where the optimal configuration is the one with the highest MRR for that task. For example, the FLT result pair consists of the FLT configuration with the highest MRR as its optimal configuration and the results of the same configuration for the DIT task. The DIT pairing is analogous. I then use the Wilcoxon signed-rank test with Holm correction to determine the statistical significance of the difference between the optimal and alternate results for any given configuration and task.

To answer *RQ 3.2*, I focus on configurations that vary the inclusion of text sources used for corpus construction. I use the same data that was collected, but the statistical analysis will differ from *RQ 3.1*. For each task, since I have paired data across all 15 configurations, i.e., each configuration has effectiveness measures for the same issues, I conduct a Friedman Chi-Square (χ^2) test. I perform post-hoc tests using Wilcoxon rank-sum test to determine which configurations have an effect.

To determine whether including a particular text source has an effect, I turn to Mann-Whitney

U tests. I compare effectiveness measures of all ranks when the text source is included against excluded. I use Mann-Whitney here because there is an unequal amount of results between a text source inclusion and exclusion due to the invalid configuration of all text sources excluded.

4. RESULTS

In this chapter, I present the results of the study described in Chapter 3. I show results for *Research Problem 1* (RP1), followed by *Research Problem 2* (RP2), and *Research Problem 3* (RP3).

4.1 Feature Location

In this section, I describe the results of the study outlined in Section 3.2.3.1.

RQ 1.1 asks how well a topic model trained on changesets performs compared to one trained on source code entities. Table 4.1 summarizes the results of each subject system when evaluated at the file-level. In the table, I bold the greater of the two MRRs. Since the goal is to show that training with changesets is just as good, or better than, training on snapshots, I 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, with respect to *RQ 1.1*, OpenJPA v2.3.0 is a favorable case for changesets as it has a higher MRR, along with statistical significance ($p < 0.01$) and a notably large effect size (0.3867). Likewise, Tika v1.8 displays a favorable case for snapshots in terms of higher MRR, but does not achieve statistical significance and is hence is not a definite unfavorable case.

I note an improvement in MRR for 4 of the 6 systems when using changesets. Mahout v0.10.0 is the only system with an MRR in favor of snapshots and statistically significant at $p < 0.01$ with the greatest effect size of all systems (0.4556). For Mahout v0.10.0, however, the difference

Table 4.1: Feature Location (*RQ 1.1*): MRR, Wilcoxon p-values, and effect size

Subject System	Successful Queries	MRR Snapshot	MRR Changesets	MRR Spread	Wilcoxon p-value	Effect size
BookKeeper v4.3.0	143	0.4510	**0.4567**	+0.0056	$p = 0.5008$	0.0758
Mahout v0.10.0	50	**0.2984**	0.2730	−0.0254	$p < 0.01$	0.4556
OpenJPA v2.3.0	131	0.2724	**0.2989**	+0.0265	$p < 0.01$	0.3867
Pig v0.14.0	174	0.3231	**0.3930**	+0.0699	$p < 0.01$	0.2858
Tika v1.8	36	**0.4778**	0.4033	−0.0744	$p = 0.4491$	0.1573
ZooKeeper v3.5.0	241	0.4742	**0.4818**	+0.0075	$p < 0.01$	0.3188
All	775	0.3907	**0.4092**	+0.0185	$p < 0.01$	0.2754

Table 4.2: Feature Location (*RQ 1.2*): MRR, Wilcoxon p-values, and effect size

Subject System	Successful Queries	MRR Batch	MRR Historical Sim.	MRR Spread	Wilcoxon p-value	Effect size
BookKeeper v4.3.0	143	**0.4567**	0.3496	−0.1070	$p < 0.01$	0.2921
Mahout v0.10.0	50	0.2730	**0.3398**	+0.0668	$p = 0.9674$	0.0081
OpenJPA v2.3.0	131	**0.2989**	0.2169	−0.0820	$p < 0.01$	0.3731
Pig v0.14.0	174	**0.3930**	0.2784	−0.1146	$p < 0.01$	0.3228
Tika v1.8	36	**0.4033**	0.3984	−0.0049	$p = 0.0375$	0.4366
ZooKeeper v3.5.0	241	**0.4818**	0.3419	−0.1398	$p < 0.01$	0.3035
All	775	**0.4092**	0.3104	−0.0987	$p < 0.01$	0.3007

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 FLT is on par with snapshot-based DIT and the better choice for the majority of systems.

RQ 1.2 asks how well a TM-based FLT 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. Table 4.2 summarizes the results of each subject system when evaluated at the file-level. In each of the tables, I bold the greater of the two MRRs. Again, since the goal is to show that temporal

Table 4.3: Developer Identification (*RQ 2.1*): MRR, Wilcoxon p-values, and effect size

Subject System	Successful Queries	MRR Snapshot	MRR Changesets	MRR Spread	Wilcoxon p-value	Effect size
BookKeeper v4.3.0	164	**0.6600**	0.6513	−0.0086	$p = 0.0134$	0.2425
Mahout v0.10.0	133	0.2374	**0.3340**	+0.0966	$p = 0.0606$	0.1998
OpenJPA v2.3.0	137	0.2728	**0.3648**	+0.0920	$p < 0.01$	0.3642
Pig v0.14.0	222	**0.2870**	0.1759	−0.1110	$p < 0.01$	0.6475
Tika v1.8	40	**0.3915**	0.3609	−0.0306	$p = 0.3119$	0.1889
ZooKeeper v3.5.0	359	**0.4542**	0.3985	−0.0557	$p = 0.6817$	0.0273
All	1055	**0.3977**	0.3770	−0.0207	$p < 0.01$	0.1218

considerations must be given during FLT evaluation, I only care about statistical significance when the MRR is in favor of batch.

There is an improvement in favor of historical simulation in MRR for only 1 of the 6 systems, which is not statistically significant at $p = 0.96$. Four of the 5 results in favor of batch changesets were statistically significant, with the last system, Tika v1.8, insignificant at $p = 0.03$. Overall, batch-trained changeset-based TMs perform better than a full historical simulation with statistical significance. This suggests that under historical simulation, the accuracy of the FLT will fluctuate as a project evolves, which may indicate a more accurate evaluation is possible with a historical simulation.

To summarize, in *RQ 1.1* I found that changeset-based FLTs can be as accurate as snapshot-based FLTs. I find in *RQ 1.2* that historical simulation reveals that the accuracy of the changeset-based FLT is inconsistent as a project evolves, and is lower than indicated by batch evaluation.

4.2 Developer Identification

In this section, I describe the results of the study outlined in Section 3.2.3.2.

RQ 2.1 asks how well a topic model trained on changesets performs compared to one trained

Table 4.4: Developer Identification (*RQ 2.2*): MRR, Wilcoxon p-values, and effect size

Subject System	Successful Queries	MRR Batch	MRR Historical Sim.	MRR Spread	Wilcoxon p-value	Effect size
BookKeeper v4.3.0	163	**0.6522**	0.5764	−0.0759	$p = 0.7518$	0.0328
Mahout v0.10.0	130	**0.3310**	0.2614	−0.0696	$p = 0.0621$	0.1978
OpenJPA v2.3.0	136	**0.3650**	0.2758	−0.0891	$p < 0.01$	0.2986
Pig v0.14.0	221	0.1722	**0.2666**	+0.0944	$p < 0.01$	0.7573
Tika v1.8	39	**0.3689**	0.3648	−0.0041	$p = 0.2402$	0.2353
ZooKeeper v3.5.0	354	**0.3923**	0.3315	−0.0607	$p < 0.01$	0.3822
All	1043	**0.3742**	0.3413	−0.0329	$p = 0.8197$	0.0089

on source code entities. Table 4.3 summarizes the results of each subject system. In the table, I bold which of the two MRRs is greater. Since the goal is to show that training with changesets is just as good, or better than, training on snapshots, I 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, with respect to *RQ 2.1*, OpenJPA v2.3.0 is a favorable case for changesets as it has a higher MRR, along with statistical significance ($p < 0.01$) and a notably large effect size (0.3642). Likewise, Tika v1.8 displays a favorable case for snapshots in terms of higher MRR, but does not achieve statistical significance and is hence is not a definite unfavorable case.

I note an improvement in MRR for only 2 of the 6 systems when using changesets. Pig v0.14.0 is the only system with an MRR in favor of snapshots that is also statistically significant at $p < 0.01$ with an effect size of 0.64. Comparing all systems at once by combining all effectiveness measures, snapshots show slight MRR improvement over changesets with statistical significance. This suggests that the changeset-based DIT is not on par with snapshot-based DIT.

RQ 2.2 asks how well a TM-based DIT would perform as it were to be used in real-time. This is a much closer evaluation of an DIT to it being used in an actual development environment. Table 4.4 summarizes the results of each subject system. In each of the tables, I bold which of the two MRRs is greater. Again, since the goal is to show that temporal considerations must be given during DIT evaluation, I only care about statistical significance when the MRR is in favor of batch.

There is an improvement in favor of historical simulation in MRR for only 1 of the 6 systems, which is statistically significant at $p < 0.01$. Two of the 5 results in favor of batch changesets were statistically significant. I note that Pig v0.14.0, in favor of snapshots over batch changesets with statistical significance for *RQ 2.1*, is statistically significant in favor of historical simulation over batch changesets for *RQ 2.2*.

Comparing all systems at once by combining all effectiveness measures, batch changesets performs better than a full historical simulation, but is not statistically significant. This suggests that under historical simulation, the accuracy of the DIT will fluctuate as a project evolves, which may indicate a more accurate evaluation is possible with a historical simulation. This aligns with the result I see for FLT in the previous section.

To summarize, in *RQ 2.1* I found changeset-based DITs are *not* as accurate as snapshot-based DITs. I find in *RQ 2.2* historical simulation reveals that the accuracy of the changeset-based DIT is inconsistent as a project evolves, and is lower than indicated by batch evaluation.

4.3 Combining and Configuring Changeset-based Topic Models

In this section, I describe the results of the study outlined in Section 3.2.3.3.

RQ 3.1 asks whether a single topic model can be used for more than a single task, specifically

for feature location and developer identification. Tables 4.5 and 4.6 show the summary results over all subject systems for model construction for the FLT and DIT tasks, respectively, while Tables 4.7 and 4.8 show the summary corpus construction for each respective task (figures and data for all subject systems are available in Appendices C and D). The goal is not to find which configuration works best for each system and task, but rather to determine whether one configuration is *capable* of producing acceptable results for both tasks.

Table 4.5: Wilcoxon test results for FLT optimal and alternate model configurations (*RQ 3.1*)

Subject	Optimal Configuration	Alternate Configuration	MRRs		p-value	Effect size
BookKeeper v4.3.0	$(K = 500, \alpha = 5/K, \eta = auto)$	$(K = 500, \alpha = 5/K, \eta = auto)$	0.4884	0.4884		
Mahout v0.10.0	$(K = 500, \alpha = 2/K, \eta = auto)$	$(K = 500, \alpha = 1/K, \eta = 5/K)$	0.3390	0.2802	0.0595	0.3474
OpenJPA v2.3.0	$(K = 500, \alpha = 2/K, \eta = auto)$	$(K = 500, \alpha = auto, \eta = 1/K)$	0.3089	0.2983	0.1182	0.1928
Pig v0.14.0	$(K = 500, \alpha = 1/K, \eta = 2/K)$	$(K = 200, \alpha = 5/K, \eta = 1/K)$	0.3964	0.2859	0.0873	0.1589
Tika v1.8	$(K = 200, \alpha = 2/K, \eta = 2/K)$	$(K = 500, \alpha = 2/K, \eta = 5/K)$	0.4831	0.3922	0.5445	0.1310
ZooKeeper v3.5.0	$(K = 500, \alpha = 1/K, \eta = auto)$	$(K = 500, \alpha = 2/K, \eta = 2/K)$	0.4882	0.4670	0.0667	0.1737
All subject systems	$(K = 500, \alpha = 2/K, \eta = auto)$	$(K = 500, \alpha = 5/K, \eta = auto)$	0.4162	0.4107	$p < 0.01$	0.2181

Table 4.6: Wilcoxon test results for DIT optimal and alternate model configurations (*RQ 3.1*)

Subject	Optimal Configuration	Alternate Configuration	MRRs		p-value	Effect size
BookKeeper v4.3.0	$(K = 500, \alpha = 5/K, \eta = auto)$	$(K = 500, \alpha = 5/K, \eta = auto)$	0.6642	0.6642		
Mahout v0.10.0	$(K = 500, \alpha = 1/K, \eta = 5/K)$	$(K = 500, \alpha = 2/K, \eta = auto)$	0.3544	0.3504	0.9324	0.0099
OpenJPA v2.3.0	$(K = 500, \alpha = auto, \eta = 1/K)$	$(K = 500, \alpha = 2/K, \eta = auto)$	0.3695	0.3466	0.0112	0.3628
Pig v0.14.0	$(K = 200, \alpha = 5/K, \eta = 1/K)$	$(K = 500, \alpha = 1/K, \eta = 2/K)$	0.2173	0.1631	0.0114	0.2070
Tika v1.8	$(K = 500, \alpha = 2/K, \eta = 5/K)$	$(K = 200, \alpha = 2/K, \eta = 2/K)$	0.3775	0.3328	0.0208	0.4617
ZooKeeper v3.5.0	$(K = 500, \alpha = 2/K, \eta = 2/K)$	$(K = 500, \alpha = 1/K, \eta = auto)$	0.4213	0.4011	0.1747	0.1045
All subject systems	$(K = 500, \alpha = 5/K, \eta = auto)$	$(K = 500, \alpha = 2/K, \eta = auto)$	0.3818	0.3718	0.2262	0.0632

Table 4.7: Wilcoxon test results for FLT optimal and alternate corpus configurations (*RQ 3.1* and *RQ 3.2*)

Subject	Optimal Configuration	Alternate Configuration	MRRs		p-value	Effect size
BookKeeper v4.3.0	(A,C,M)	(A,R,C,M)	0.5327	0.5246	0.7311	0.0410
Mahout v0.10.0	(C,M)	(A)	0.3650	0.2601	0.3049	0.1827
OpenJPA v2.3.0	(A,C,M)	(A,C)	0.3687	0.2869	0.1908	0.1425
Pig v0.14.0	(A,R)	(C)	0.4360	0.3124	$p < 0.01$	0.2529
Tika v1.8	(A,R,C,M)	(A,C)	0.6482	0.5572	0.3943	0.2165
ZooKeeper v3.5.0	(A,C,M)	(C)	0.4950	0.3930	$p < 0.01$	0.2400
All subject systems	(A,C,M)	(C,M)	0.4517	0.3971	$p < 0.01$	0.1273

Table 4.8: Wilcoxon test results for DIT optimal and alternate corpus configurations (*RQ 3.1* and *RQ 3.2*)

Subject	Optimal Configuration	Alternate Configuration	MRRs		p-value	Effect size
BookKeeper v4.3.0	(A,R,C,M)	(A,C,M)	0.7216	0.6974	$p < 0.01$	0.7783
Mahout v0.10.0	(A)	(C,M)	0.3827	0.3413	0.8252	0.0238
OpenJPA v2.3.0	(A,C)	(A,C,M)	0.4096	0.3935	0.5454	0.0687
Pig v0.14.0	(C)	(A,R)	0.3032	0.1437	$p < 0.01$	0.8348
Tika v1.8	(A,C)	(A,R,C,M)	0.4522	0.3943	0.4990	0.1569
ZooKeeper v3.5.0	(C)	(A,C,M)	0.4565	0.3667	$p < 0.01$	0.3818
All subject systems	(C,M)	(A,C,M)	0.4165	0.3785	$p < 0.01$	0.2580

When I consider all systems together for model construction (Tables 4.5 and 4.6), the both tasks perform best when $K = 500$ and $\eta = auto$, but differ when it comes to α. FLT prefers the lower alpha, $2/K$, while DIT is best with the highest alpha, $5/K$. A Wilcoxon rank-sum test reveals that the optimal model for FLT ($K = 500$, $\alpha = 2/K$, $\eta = auto$)is statistically significant ($p < 0.01$) compared to the alternate with an effect size low at $r = 0.2181$. Likewise, a Wilcoxon rank-sum test reveals that the optimal model for DIT ($K = 500$, $\alpha = 5/K$, $\eta = auto$)is not statistically significant ($p = 0.2262$) compared to the alternate. This suggests that the DIT is less sensitive to parameter changes, allowing the user to select the model configuration that works best for FLT.

When I consider all systems together for corpus construction (Tables 4.7 and 4.8), the both tasks perform best when including the context and message, but differ when it comes to including additions. FLT prefers additions to be included, while DIT does not. Neither task performs best when removals are included. A Wilcoxon rank-sum test reveals that the optimal corpus for FLT (additions, context, message)is statistically significant ($p < 0.01$) compared to the alternate with an effect size of $r = 0.1273$. Likewise, a Wilcoxon rank-sum test also reveals that the optimal corpus for DIT (context, message)is statistically significant ($p < 0.01$) compared to the alternate with an effect size of $r = 0.2580$. This suggests that correct corpus construction plays a somewhat important role in both tasks, but the effect of choosing an alternate over the optimal is low.

RQ 3.2 asks what portions of a changeset are most critical for performance of different tasks, specifically for feature location and developer identification. As in *RQ 3.1*, the goal is not to find which configuration works best for each system and task, but rather to determine whether a particular text source affects the tasks. Tables 4.9 and 4.10 show the Friedman test results for FLT and DIT, respectively.

Table 4.9: Friedman test results for FLT corpus configuration sweeps (*RQ 3.2*). For each system, 105 post-hoc Wilcoxon tests were conducted.

Subject	$\chi^2(15)$	p-value	Post-hoc Wilcoxon
BookKeeper v4.3.0	120.1787	$p < 0.01$	37 (35.2%)
Mahout v0.10.0	13.5958	0.4802	0 (0.0%)
OpenJPA v2.3.0	39.3602	$p < 0.01$	8 (7.6%)
Pig v0.14.0	53.3693	$p < 0.01$	12 (11.4%)
Tika v1.8	39.1413	$p < 0.01$	9 (8.6%)
ZooKeeper v3.5.0	71.6131	$p < 0.01$	26 (24.8%)
All subject systems	204.9252	$p < 0.01$	49 (46.7%)

Table 4.10: Friedman test results for DIT corpus configuration sweeps (*RQ 3.2*). For each system, 105 post-hoc Wilcoxon tests were conducted.

Subject	$\chi^2(15)$	p-value	Post-hoc Wilcoxon
BookKeeper v4.3.0	269.1597	$p < 0.01$	57 (54.3%)
Mahout v0.10.0	50.5514	$p < 0.01$	23 (21.9%)
OpenJPA v2.3.0	124.6299	$p < 0.01$	42 (40.0%)
Pig v0.14.0	887.0940	$p < 0.01$	85 (81.0%)
Tika v1.8	23.1535	0.0578	1 (1.0%)
ZooKeeper v3.5.0	454.4956	$p < 0.01$	80 (76.2%)
All subject systems	1026.4202	$p < 0.01$	86 (81.9%)

As seen in the results for *RQ 3.1*, the results for the corpus construction sweep vary across systems and tasks. Importantly, there does not seem to be any particular configuration that is shared between systems or tasks. For all subject systems, both tasks perform best when including changeset context and message text sources. FLT sees a slight increase in MRR (0.0546) when the additions text source is included, while DIT slightly decreases in MRR (0.038).

The Friedman test of all subject systems shows significance for both FLT ($p < 0.01$) and DIT ($p < 0.01$), indicating a difference between configurations for each task. Likewise, the Friedman test for each individual system shows significance for both FLT ($p < 0.01$) and DIT ($p < 0.01$), with two exceptions. The two non-significant exceptions were Mahout v0.10.0 FLT task with $p = 0.4802$ and Tika v1.8 DIT task with $p = 0.0578$.

A post-hoc test was conducted using Wilcoxon rank-sum test with Holm correction for each possible configuration pairing (105 tests). For combined effectiveness measures of all subject systems, 49 pairs were statistically significant at $p < 0.01$ for the FLT task. Similarly, for the DIT task, combined results of all subject systems contains 86 pairs that are statistically significant at $p < 0.01$.

To summarize, in *RQ 3.1* I found that the same topic model can be used in more than one context, though more optimal configurations may exist on a per-context basis. In *RQ 3.2* I find that there are significant differences when choosing from the possible elements of a changeset for corpus construction.

5. DISCUSSION

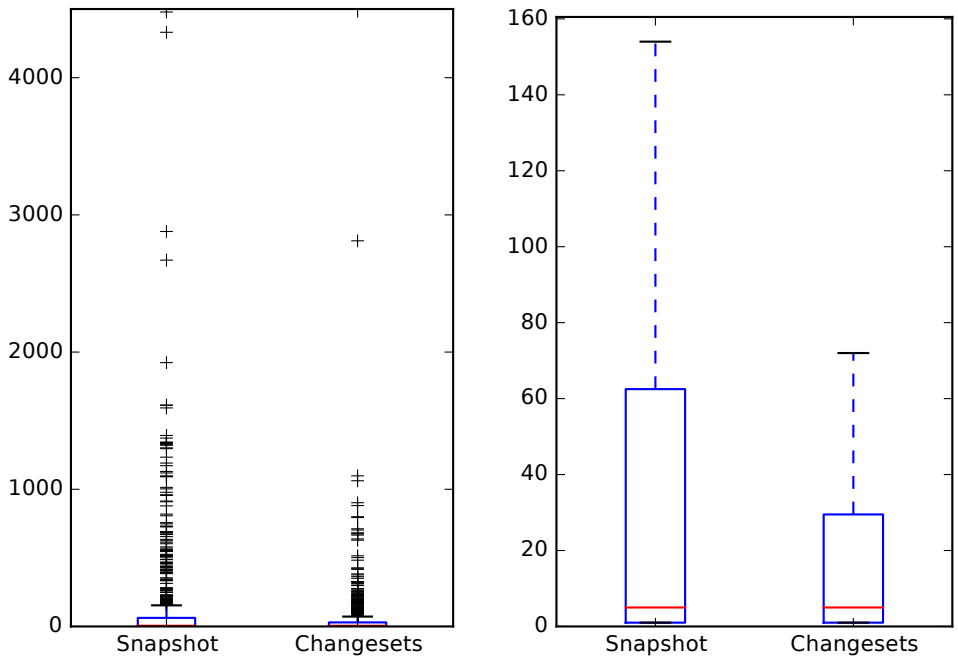
In this chapter, I will take a more qualitative look at the Results of the study described in Chapter 4. I discuss the findings for each research question, starting with *RQ 1.1* and *RQ 1.2* (RP1), followed by *RQ 2.1* and *RQ 2.2* (RP2), and then finally *RQ 3.1* and *RQ 3.2* (RP3). I will then present a summary of the findings across all three research problems. Finally, I discuss the threats to validity of the study and my attempts to mitigate them.

5.1 Feature Location

The analysis in Section 4.1 shows significant effects between snapshots and changesets, and between batch changesets and changesets in the simulated environment. The results are mixed between each subject system and are not generally conclusive. However, I argue this is desirable to show that the accuracy of a changeset-based FLT is similar to that of a snapshot-based FLT but without the retraining cost.

5.1.1 Is a changeset-based FLT as accurate as a snapshot-based FLT?

Figure 5.1 shows the effectiveness measures for files across all systems (figures for all subject systems are available in Appendix A). The figure suggests that snapshot-based models and changeset-based models have similar results overall with changesets performing slightly better, but does not help to understand how each feature query performs for each model. With respect to *RQ 1.1*, I will investigate the queries and effectiveness measures between the batch snapshot and batch changesets in detail.

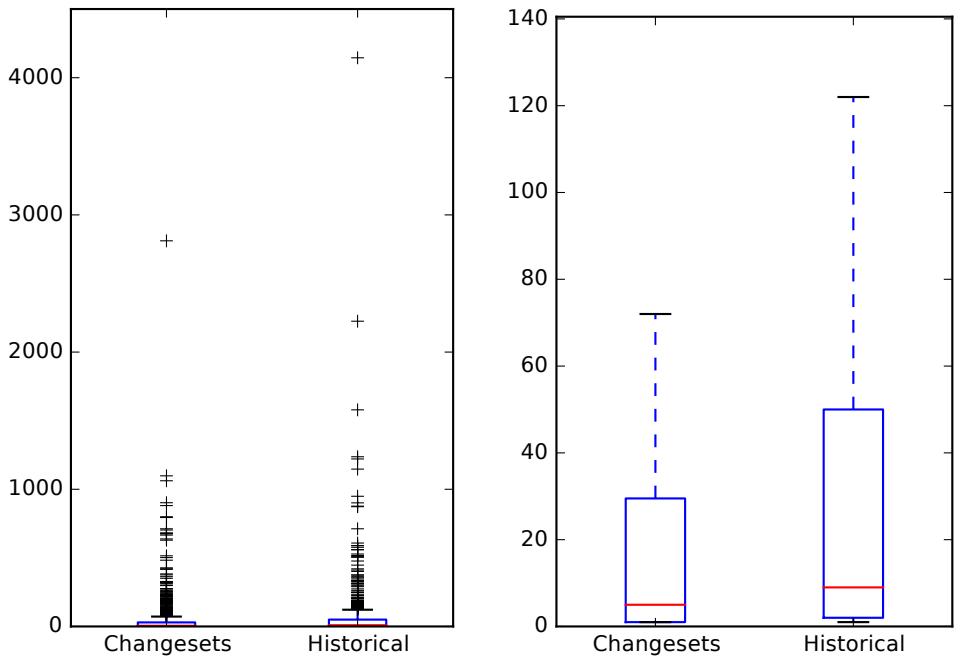


(a) Including outliers (b) Excluding outliers

Figure 5.1: *RQ 1.1*: Feature Location effectiveness measures for all subject systems

For the 1055 queries across all systems, only 775 were successful. That is, 280 queries did not retrieve any files identified as changed by fixing the related issues. These are likely caused by files that were removed over time and did not make it into release and highlights the volatility of software development. It is typical for unsuccessful queries to be filtered from the benchmark, hence I have removed them from the data analyses instead of penalizing a particular approach.

There were 155 queries that return the same effectiveness measure in both approaches, or about 20.0% of the time. Of these 155 queries, 139 (17.9%) have an effectiveness measure of 1 (the best possible measure) for both approaches. After excluding the 155 queries in which ranks which are the same, 87 (11.2%) of the remaining 620 queries have effectiveness measures is within 1 rank of each other. Likewise, 254 (32.8%) queries have a difference in effectiveness measure less than or



(a) Including outliers (b) Excluding outliers

Figure 5.2: *RQ 1.2*: Feature Location effectiveness measures for all subject systems

equal to 10. Finally, 398 (51.4%) have the effectiveness measure is within 50 ranks of each other. The remaining 222 queries (28.6%) perform noticeably different ($>$ 50 ranks apart).

5.1.2 Does the accuracy of a changeset-based FLT fluctuate as a project evolves?

Figure 5.2 shows the effectiveness measures for files across all systems (figures for all subject systems are available in Appendix A). The figure suggests that historical simulation slightly underperforms a batch changeset model, but does not help to understand how each feature query performs for each model. With respect to *RQ 1.2*, I will investigate the queries and effectiveness measures between the batch changesets and historical simulation using changesets in detail.

For the 1055 queries across all systems, again only 775 were successful. 121 queries return the same effectiveness measure in both approaches, or about 15.6% of the time. Of these 121 queries, 98 (12.6%) have an effectiveness measure of 1 (the best possible measure) for both approaches.

After excluding the 121 queries in which ranks which are the same, 76 (9.8%) of the remaining 654 queries have effectiveness measures is within 1 rank of each other. Likewise, 285 (36.8%) queries have a difference in effectiveness measure less than or equal to 10. Finally, 460 (59.4%) have the effectiveness measure is within 50 ranks of each other. The remaining 194 queries (25.0%) perform noticeably different ($>$ 50 ranks apart).

5.1.3 Situations

In this study, I've asked two research questions which lead to two distinct comparisons. First, I compare a batch TM-based FLT trained on the changesets of a project's history to one trained on the snapshot of source code entities. Second, I compare a batch TM-based FLT trained on changesets to a online TM-based FLT trained on the same changesets over time. The results are mixed between the research questions, hence I end up with four possible situations; I will now discuss each of these situations in detail.

5.1.3.1 Batch changesets are better than batch snapshot and *batch changesets are better than changesets in the simulated environment*

This situation occurs in 4 out of 6 systems: BookKeeper v4.3.0, OpenJPA v2.3.0, Pig v0.14.0, ZooKeeper v3.5.0. I hypothesize that this is because in the batch evaluation, the model is trained on all data before being queried, while in the historical simulation the model is trained on partial data before being queried. This allows for the batch model to be more accurate because it is trained on more data and reveals feature location research evaluations may not be accurately portraying how an FLT would perform in a real scenario. I note that this is the scenario that occurs the most and aligns with the statistical results.

5.1.3.2 *Batch changesets are better than batch snapshot* and *changesets in the simulated environment are better than batch changesets*

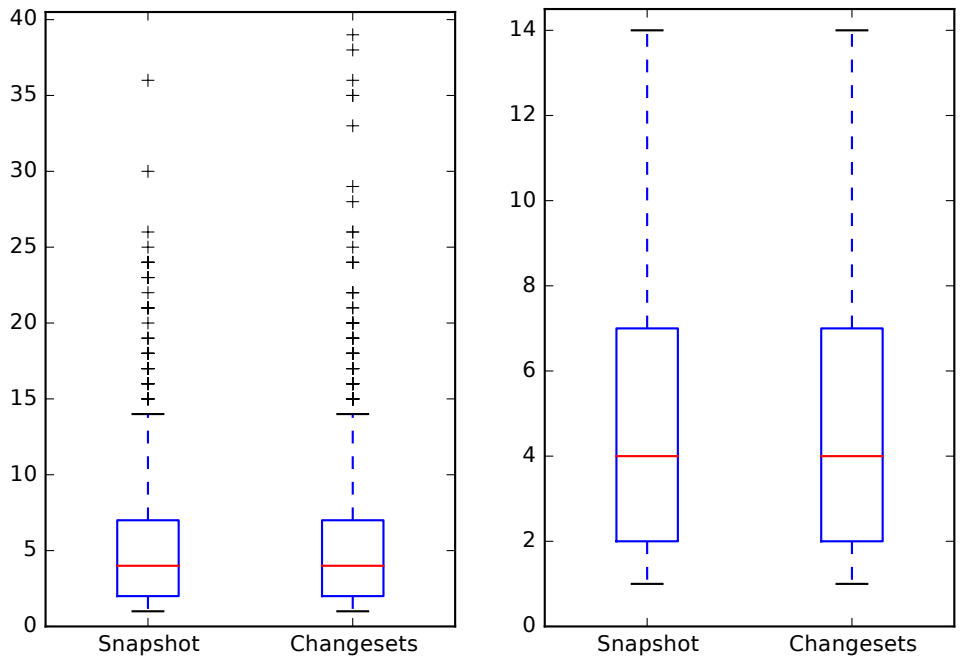
I note that this situation never occurs. This also supports the hypothesis that historical simulation more accurately portrays the system over time as it more accurately captures the correct state of the system (i.e., the source code entities) at the point in time when querying is done. Since querying on the batch models is after the model is trained, there may be source code entities that do not exist in the system anymore that were at one time changed to complete a certain task. Again, the historical simulation better captures this scenario.

5.1.3.3 *Batch snapshot are better than batch changeset* and *changesets in the simulated environment are better than batch changesets*

This situation occurs in a single system: Mahout v0.10.0. Similarly, this could be because of how the models are trained. Although batch changesets perform worse in both cases, historical simulation using changesets outweighs batch snapshot modelling. This does not necessarily mean that changesets are bad, but may more accurately model the system over time.

5.1.3.4 *Batch snapshot are better than batch changeset* and *batch changesets are better than changesets in the simulated environment*

This situation occurs in a single system: Tika v1.8. I note that this system does not achieve statistical significance for either case. This also supports the hypothesis that historical simulation more accurately portrays the system over time. However, I cannot conclude this without also historically simulating snapshot TM-based FLTs.



(a) Including outliers (b) Excluding outliers

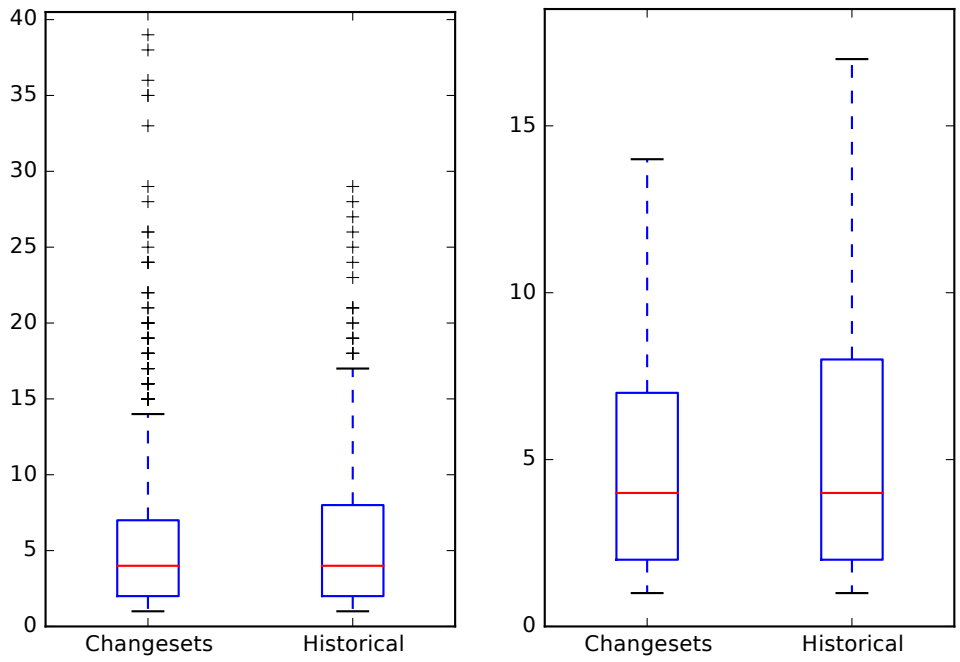
Figure 5.3: *RQ 2.1*: Developer Identification effectiveness measures for all subject systems

5.2 Developer Identification

The results Section 4.2 shows significant effects between snapshots and changesets, and between batch changesets and changesets in the simulated environment. The results are mixed between each and are not conclusive. However, I argue this is desirable to show that the accuracy of a changeset-based DIT is similar to that of a snapshot-based DIT but without the retraining cost.

5.2.1 Is a changeset-based DIT as accurate as a snapshot-based DIT?

Figure 5.3 shows the effectiveness measures across all systems (figures for all subject systems are available in Appendix B). The figure suggests that snapshot-based models and changeset-based models have similar results overall with changesets performing slightly better, but does not help to understand how each feature query performs for each model. With respect to *RQ 2.1*, I will



(a) Including outliers (b) Excluding outliers

Figure 5.4: *RQ 2.2*: Developer Identification effectiveness measures for all subject systems

investigate the queries and effectiveness measures between the batch snapshot and batch changesets in detail.

For the 1055 queries across all systems, 154 queries return the same effectiveness measure in both approaches, or about 14.6% of the time. Of these 154 queries, 58 (5.5%) share an effectiveness measure of 1 (the best possible measure) for both approaches. After excluding the 154 queries in which ranks which are the same, 245 (23.2%) of the remaining 901 queries have effectiveness measures is within 1 rank of each other. Likewise, 145 (13.8%) queries have a difference in effectiveness measure of 2. Finally, 166 (15.7%) have the effectiveness measure difference of 3. The remaining 345 queries (38.3%) perform noticeably different ($>$ 3 ranks apart).

5.2.2 Does the accuracy of a changeset-based DIT fluctuate as a project evolves?

Figure 5.4 shows the effectiveness measures across all systems (figures for all subject systems are available in Appendix B). The figure suggests that snapshot-based models and changeset-based models have similar results overall with changesets performing slightly better, but does not help to understand how each feature query performs for each model. With respect to *RQ 2.2*, I will investigate the queries and effectiveness measures between the batch snapshot and batch changesets in detail.

For the 1055 queries across all systems, only 1043 were successful. After manual investigation, I determined these 12 that were unsuccessful due to excluding the commit that fixed the issue related to the query. These commits were the first change in the repository by that maintainer[8], resulting in that maintainer missing from the ranking until that point. 126 queries return the same effectiveness measure in both approaches, or about 12.1% of the time. Of these 126 queries, 45 (4.3%) share an effectiveness measure of 1 (the best possible measure) for both approaches.

After excluding the 126 queries in which ranks which are the same, 213 (20.4%) of the remaining 917 queries have effectiveness measures is within 1 rank of each other. Likewise, 169 (16.2%) queries have a difference in effectiveness measure of 2. Finally, 130 (12.5%) have the effectiveness measure difference of 3. The remaining 512 queries (55.8%) perform noticeably different (> 3 ranks apart).

5.2.3 Situations

In this study, I've asked two research questions which lead to two distinct comparisons. First, I compare a batch TM-based DIT trained on the changesets of a project's history to one

[8] Or an alias of that developer, e.g., committed under a different email

trained on the snapshot of source code entities. Second, I compare a batch TM-based DIT trained on changesets to a online TM-based DIT trained on the same changesets over time. My results are mixed between the research questions, hence I end up with four possible situations; I will now discuss each of these situations in detail.

5.2.3.1 *Batch changesets are better than batch snapshot* and *batch changesets are better than changesets in the simulated environment*

This situation occurs in 2 out of 6 systems: Mahout v0.10.0 and OpenJPA v2.3.0. I hypothesize that this is because in the batch evaluation, the model is trained on all data before being queried, while in the historical simulation the model is trained on partial data before being queried. This allows for the batch model to be more accurate because it is trained on more data and reveals feature location research evaluations may not be accurately portraying how an DIT would perform in a real scenario.

5.2.3.2 *Batch changesets are better than batch snapshot* and *changesets in the simulated environment are better than batch changesets*

I note that this situation never occurs. This also supports the hypothesis that historical simulation more accurately portrays the system over time as it more accurately captures the correct state of the system (i.e., the developers maintaining the system) at the point in time when querying is done. Since querying on the batch models is after the model is completely trained, there may be developers that are no longer, or have yet to start, maintaining the system anymore. Again, the historical simulation may better capture this scenario.

5.2.3.3 *Batch snapshot are better than batch changeset* and *changesets in the simulated environment are better than batch changesets*

This situation occurs in 3 out of 6 systems: BookKeeper v4.3.0, Tika v1.8, and ZooKeeper v3.5.0. Similarly, this could be because of how the models are trained. Although batch changesets perform worse in both cases, historical simulation using changesets outweighs batch snapshot modelling. This does not necessarily mean that changesets are bad, but may more accurately model the system over time.

5.2.3.4 *Batch snapshot are better than batch changeset* and *batch changesets are better than changesets in the simulated environment*

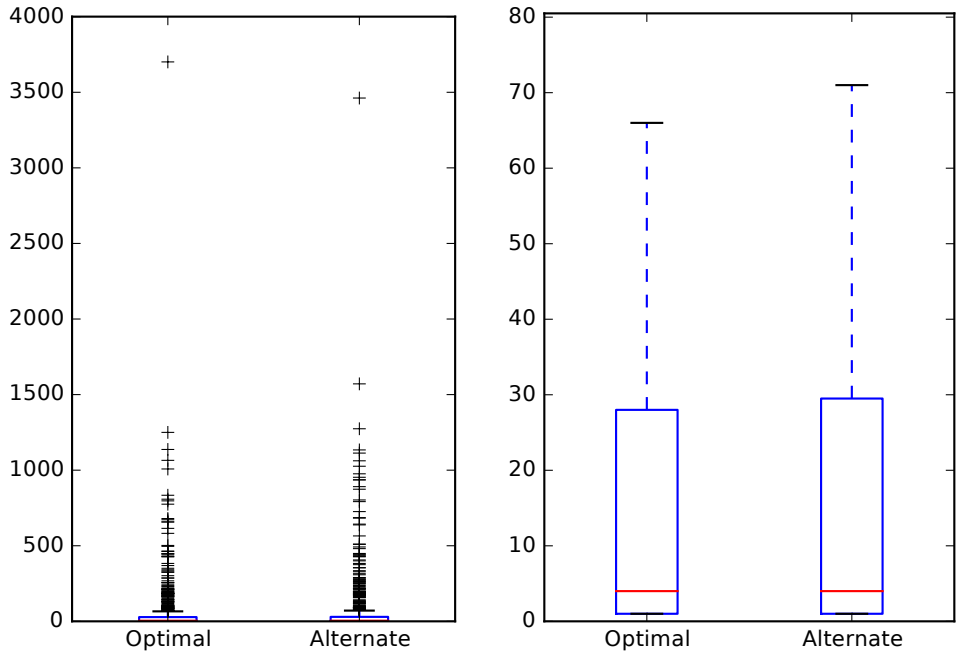
This situation occurs in a single system: Pig v0.14.0. I note that this system does not achieve statistical significance for either case. This also supports the hypothesis that historical simulation more accurately portrays the system over time. However, I cannot conclude this without also historically simulating snapshot TM-based DITs.

5.3 Combining and Configuring Changeset-based Topic Models

The results for *Research Problem 3* (RP3) in Section 4.3 shows significant effects between different configurations of the topic model and corpus construction.

5.3.1 Can we use the same topic model in more than one context effectively?

In *RQ 3.1*, I ask whether it is feasible to use a single model for two tasks, FLT and DIT. I explore this question from two directions: the input to the model, i.e., corpus construction, and the model configuration itself. Each of these directions contains choices that every application must



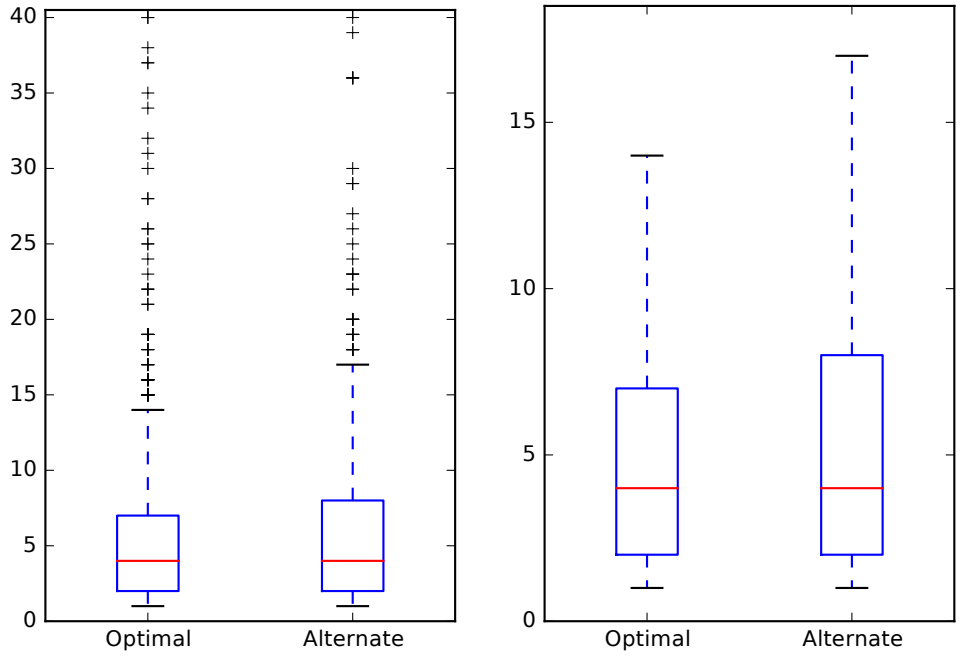
(a) Including outliers (b) Excluding outliers

Figure 5.5: Feature Location effectiveness measures of optimal ($MRR = 0.4162$) and alternate ($MRR = 0.4107$) model configurations for all subject systems

make to train an optimal model for the application's task. However, to choose an optimal model for two tasks, then some trade-offs must be considered and made.

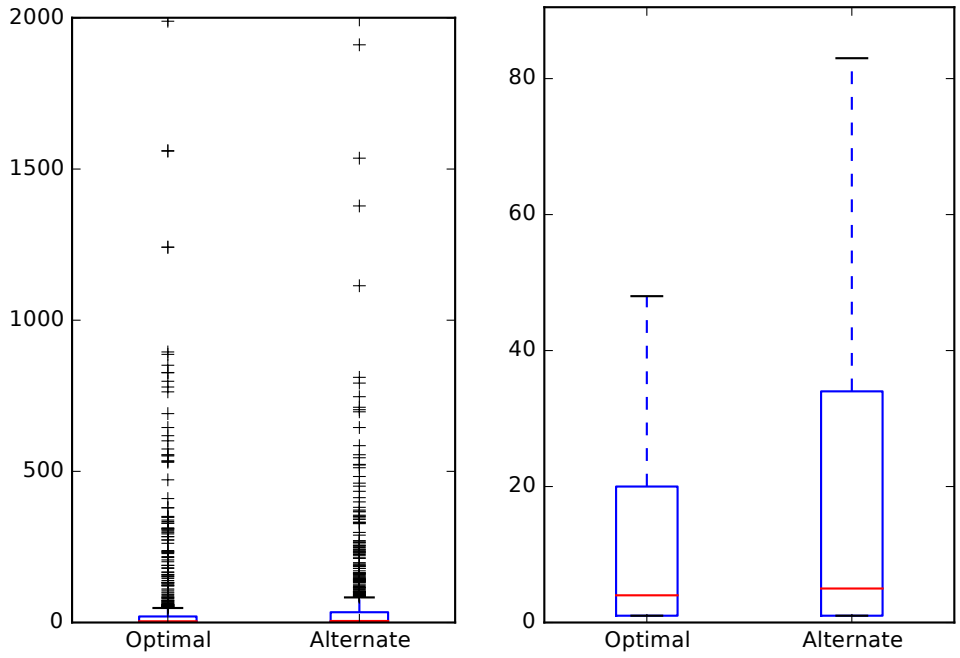
Figures 5.5 and 5.6 show the effectiveness measures for across all systems of the optimal and alternate model configuration of the FLT and DIT tasks, respectively (figures and data for individual subject systems are available in Appendix C). Likewise, Figure 5.7 and 5.8 show the effectiveness measures for across all systems of the optimal and alternate corpus construction of the FLT and DIT tasks, respectively (figures and data for individual subject systems are available in Appendix D).

With respect to model configuration, some interesting results can be found. For example, BookKeeper v4.3.0 has the same optimal configuration for both tasks, so no trade-offs must be made. Mahout v0.10.0 varied the most for FLT with the highest effect size, but has the least difference



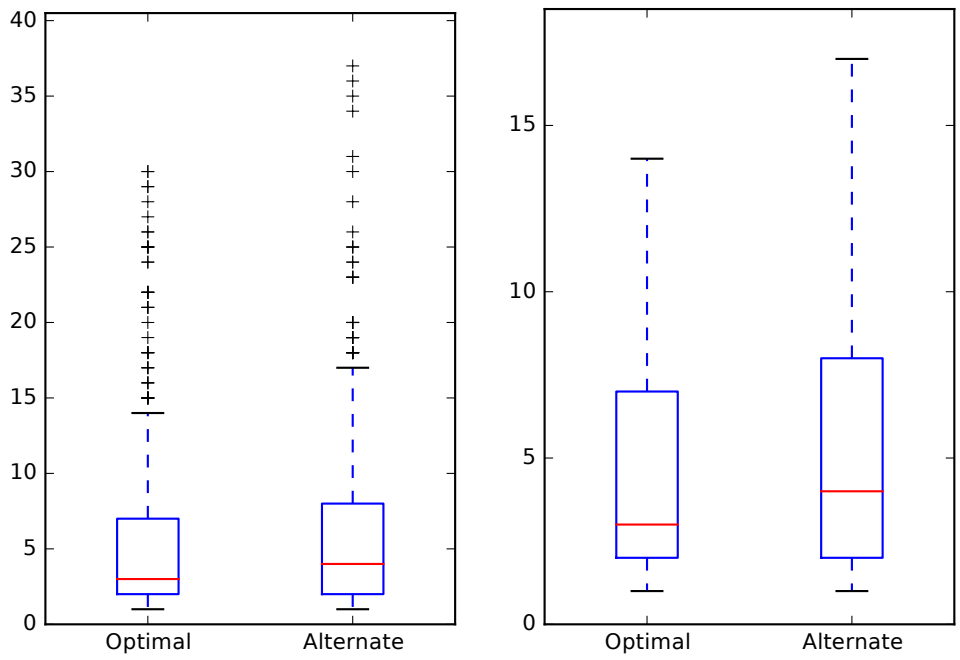
(a) Including outliers (b) Excluding outliers

Figure 5.6: Developer Identification effectiveness measures of optimal ($MRR = 0.3818$) and alternate ($MRR = 0.3718$) model configurations for all subject systems



(a) Including outliers (b) Excluding outliers

Figure 5.7: Feature Location effectiveness measures of optimal ($MRR = 0.4517$) and alternate ($MRR = 0.3971$) corpus configurations for all subject systems



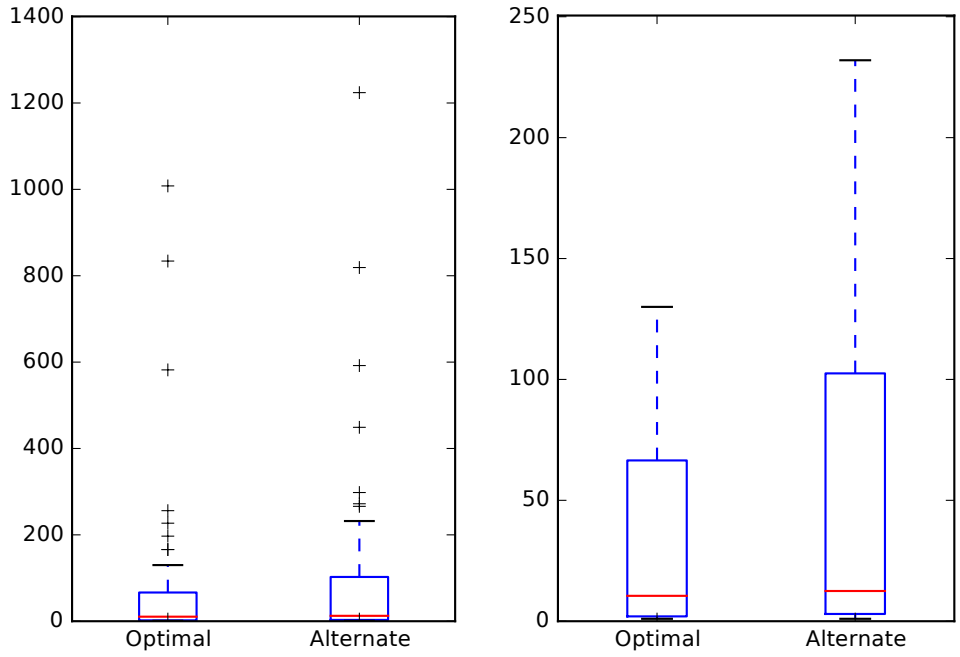
(a) Including outliers (b) Excluding outliers

Figure 5.8: Developer Identification effectiveness measures of optimal ($MRR = 0.4165$) and alternate ($MRR = 0.3785$) corpus configurations for all subject systems

for DIT, indicating that the application should defer to the FLT configuration as DIT does not significantly suffer from this decision. Figures 5.9 and 5.10 show the effectiveness measures of Mahout v0.10.0 and showcase a unique property: if outliers were to be excluded, the performance of DIT would be in favor of the alternate configuration, i.e., both tasks could have the same optimal configuration.

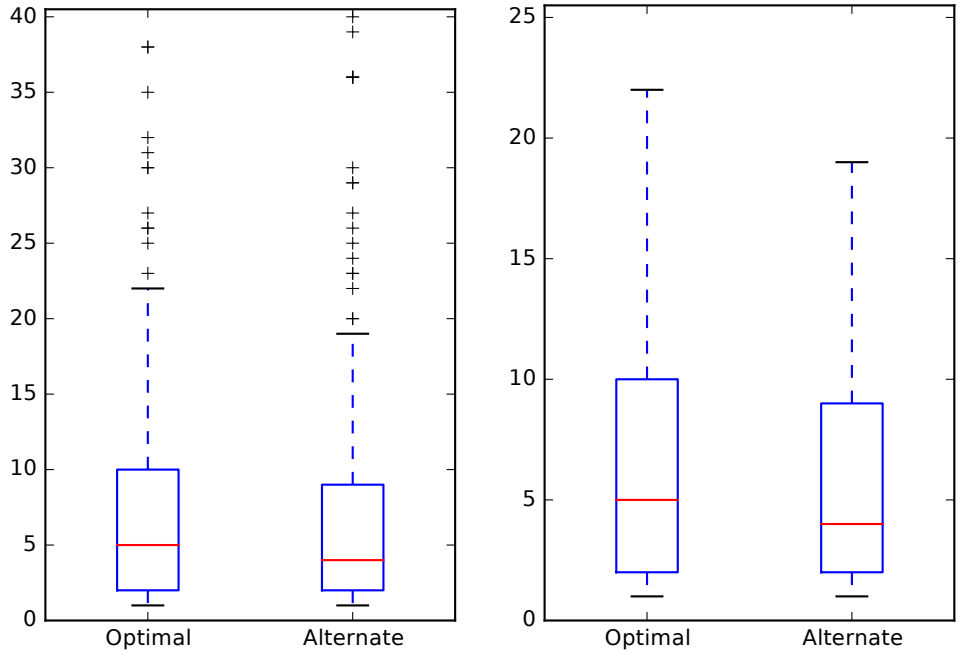
Likewise, OpenJPA v2.3.0 varied the most for DIT with the highest effect size, but did not have large effect size for FLT, again with the alternate FLT not significantly suffering from choosing the optimal DIT configuration. Figures 5.11 and 5.12 show the effectiveness measures of OpenJPA v2.3.0 and showcase's the same property as Mahout v0.10.0, but for FLT performance.

For all subject systems, however, I do have a statistical significant result when comparing



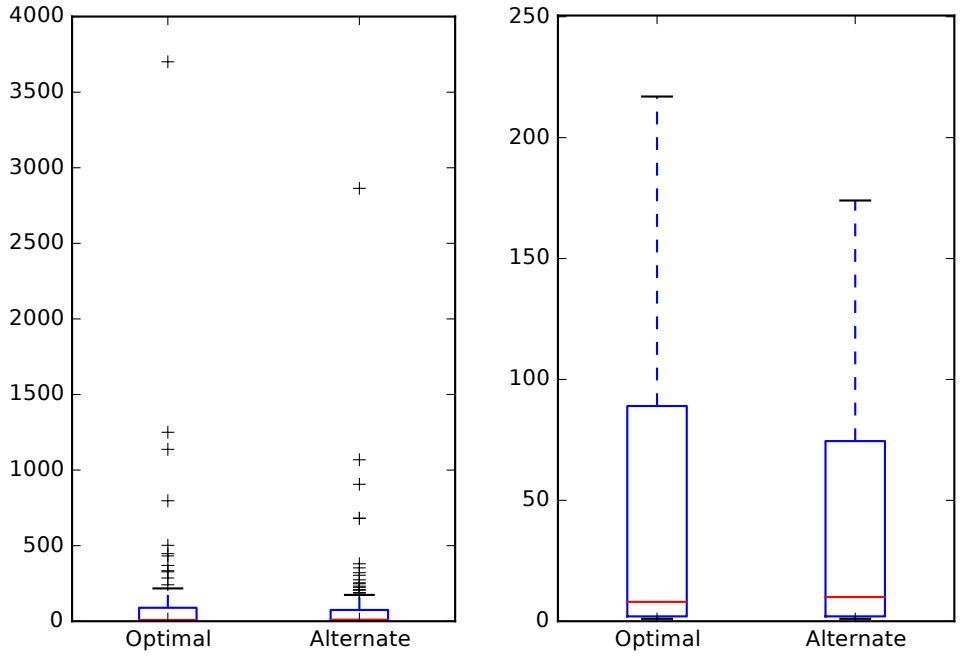
(a) Including outliers (b) Excluding outliers

Figure 5.9: Feature Location effectiveness measures of optimal ($MRR = 0.3390$) and alternate ($MRR = 0.2802$) model configurations for Mahout v0.10.0



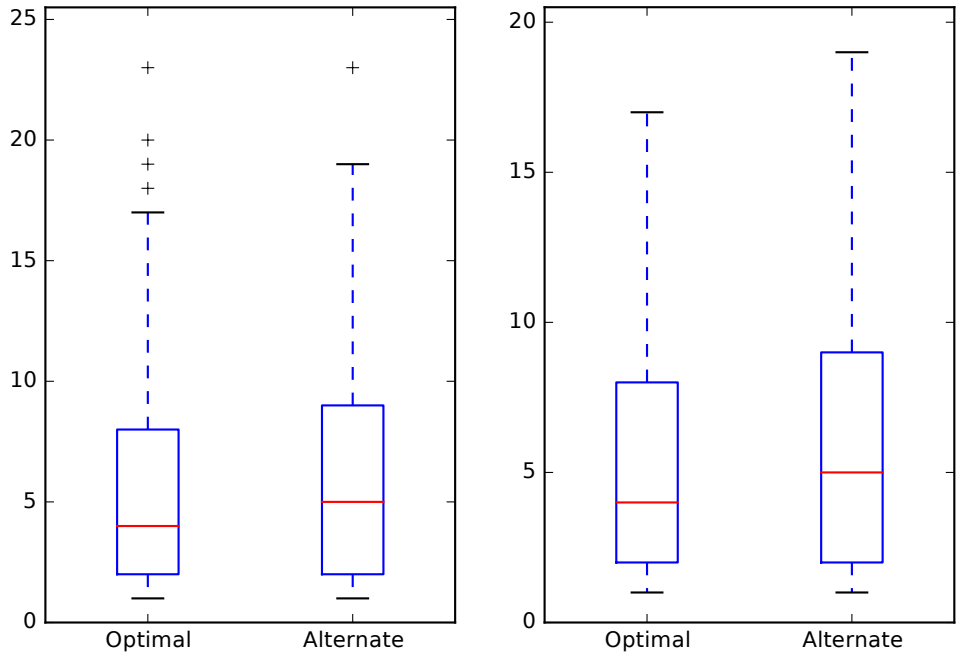
(a) Including outliers (b) Excluding outliers

Figure 5.10: Developer Identification effectiveness measures of optimal ($MRR = 0.3544$) and alternate ($MRR = 0.3504$) model configurations for Mahout v0.10.0



(a) Including outliers (b) Excluding outliers

Figure 5.11: Feature Location effectiveness measures of optimal ($MRR = 0.3089$) and alternate ($MRR = 0.2983$) model configurations for OpenJPA v2.3.0



(a) Including outliers (b) Excluding outliers

Figure 5.12: Developer Identification effectiveness measures of optimal ($MRR = 0.3695$) and alternate ($MRR = 0.3466$) model configurations for OpenJPA v2.3.0

the optimal FLT to it's alternate configuration (DIT optimal). However, the effect size is not large and the difference in MRR insignificant, indicating there are some queries which perform well in the optimal FLT configuration but not in the alternate FLT configuration, and vice versa. Interestingly, the configurations only vary by the choice for α, while number of topics (K) and η are equal.

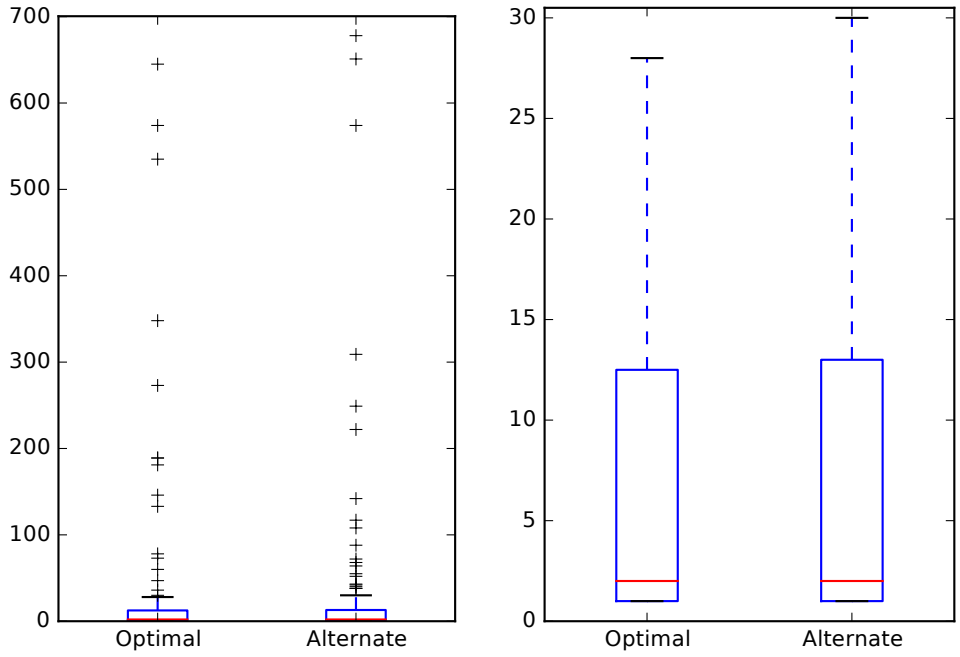
With respect to corpus construction, the choice of source text becomes more critical than model parameters. For two systems, Mahout v0.10.0 and Pig v0.14.0, the configurations have no overlapping sources, e.g., Mahout v0.10.0 optimal FLT includes context and messages, while optimal DIT only includes additions. For Pig v0.14.0 and ZooKeeper v3.5.0, there does not seem to be a easy choice as both tasks will lose significant amounts of performance if one task is preferred over the other.

Surprisingly, BookKeeper v4.3.0 had the most interesting result, with an insignificant difference between the FLT optimal and alternate, while DIT performance had significant impact with a high effect size. Figures 5.13 and 5.14 show the effectiveness measures of BookKeeper v4.3.0, of which shows little to no outliers for DIT. The two configurations only vary by exclusion of line removals, and seem to advocate for their exclusion for best performance.

5.3.2 What are the effects of using different portions of a changeset for corpus construction, such as added, removed, context lines, and the commit message?

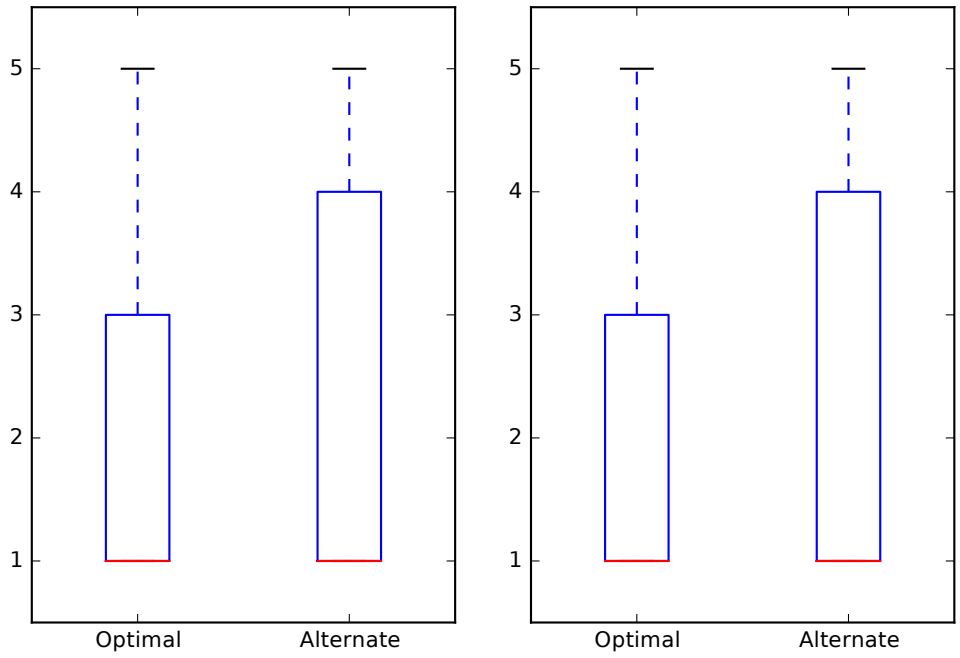
As I saw in *RQ 3.1*, it seems feasible to use a single model for two tasks, but the choice of corpus construction seems most critical. In *RQ 3.2*, I explore further just how critical those choices may be. Indeed, I see in the results in Section 4.3 that choice of text source is significant in all but one subject system for each task, or two insignificant results.

The changeset corpus sweep was completed on four elements: additions (A) context (C),



(a) Including outliers (b) Excluding outliers

Figure 5.13: Feature Location effectiveness measures of optimal ($MRR = 0.5327$) and alternate ($MRR = 0.5246$) corpus configurations for BookKeeper v4.3.0



(a) Including outliers (b) Excluding outliers

Figure 5.14: Developer Identification effectiveness measures of optimal ($MRR = 0.7216$) and alternate ($MRR = 0.6974$) corpus configurations for BookKeeper v4.3.0

Table 5.1: MRR values of all subject systems corpus construction sweep (*RQ 3.1* and *RQ 3.2*)

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

messages (M), and removals (R). Figure 1.1 shows these four elements. Intuitively, it makes sense to always include additions, as they represent the new words (code) that represent the code that was being worked on at that moment. Likewise, the 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 may not make sense, however, to include both context and removals during corpus construction as they would increase noise of certain words, e.g., over valuing words that have already appeared in additions over and over again, as in context words, and duplicating value of words that have already appeared in additions when they are no longer valid, as in removed words.

To gain insight into whether a particular source is beneficial, or detrimental, to performance of a task, I can compare MRRs of configurations that include a source to the same configuration

Table 5.2: Wilcoxon test results for additions inclusion and exclusion configurations of the FLT task for all subject systems (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(R,C,M)	**0.4315**	0.4013	$p < 0.01$	0.1366
(A,C,M)	(C,M)	**0.4517**	0.3971	$p < 0.01$	0.1273
(A,R,C)	(R,C)	**0.4092**	0.3443	$p < 0.01$	0.1739
(A,C)	(C)	**0.4010**	0.3386	0.1626	0.0637
(A,R,M)	(R,M)	**0.4119**	0.3488	$p < 0.01$	0.1825
(A,M)	(M)	**0.4147**	0.3838	0.8570	0.0082
(A,R)	(R)	**0.4214**	0.3151	$p < 0.01$	0.3554

without that particular text source, e.g., for the additions text source, I can compare configuration (A,R,C), which includes additions, removals, and context, to configuration (R,C), which includes removals and context, but excludes additions. I use this comparison, in conjunction with the mean reciprocal rank (MRR) and the Wilcoxon signed-rank test to explore the various text sources. Table 5.1 summarizes all configurations and their MRRs for each task of all subject systems. With the example, I can see that configuration (A,R,C) outperforms configuration (R,C) for both tasks. Note that for configurations with a singular text source, such as (A) for additions, there can be no comparison made because the exclusion of additions leaves me with the empty corpus.

Below I discuss these comparisons for both the FLT and DIT tasks. Each includes a discussion concerning the comparisons as applied to issues across all subject systems. Appendix E contains comparisons tables for each individual subject system.

5.3.2.1 Additions

For all seven possible configuration pairs, as shown in Table 5.2, additions improve the MRR of all configurations for FLT. Two of these results, (A,C) and (A,M) are statistically insignificant using Wilcoxon and the effect sizes of the significant five are low, ranging in the teens. The

Table 5.3: Wilcoxon test results for additions inclusion and exclusion configurations of the DIT task for all subject systems (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(R,C,M)	**0.3784**	0.3391	$p < 0.01$	0.4174
(A,C,M)	(C,M)	0.3785	**0.4165**	$p < 0.01$	0.2580
(A,R,C)	(R,C)	**0.3770**	0.3373	$p < 0.01$	0.3923
(A,C)	(C)	0.3802	**0.4148**	$p < 0.01$	0.1686
(A,R,M)	(R,M)	**0.3646**	0.3308	$p < 0.01$	0.4145
(A,M)	(M)	**0.3537**	0.3359	0.1918	0.0498
(A,R)	(R)	**0.3462**	0.3147	$p < 0.01$	0.4047

comparison with removals, (A,R) and (R), had a notable effect size that nearly doubles the next largest effect size. This suggests that including additions in an FLT is not harmful and may show gains in performance, even when paired with potentially detrimental sources, such as removals (see following subsection 5.3.2.4).

Table 5.3 shows DIT performance addition inclusion degrades under two configurations: the first, (A,C,M), being paired with the optimal DIT configuration (C,M), and the second, (A,C), underperforming the corpus only containing context (C). Six of these configurations were significant with effect sizes varying between low (0.1686) and moderate (0.4174). I note that the top four largest effect sizes were in favor of including additions. This suggests that including additions in an DIT is likely beneficial, but the choice is not necessarily straight-forward and caution is advised.

5.3.2.2 Context

In Table 5.4, I see five out of seven configuration comparisons favoring context inclusion for the FLT task. Strikingly, only one out of all comparisons was statistically significant, (C,M) compared to (M). Two configurations, (A,R,C) and (A,C), degrade in performance for the FLT task while all other configurations see improvements. For these two, the results were both non-significant

Table 5.4: Wilcoxon test results for context inclusion and exclusion configurations of the FLT task for all subject systems (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,R,M)	**0.4315**	0.4119	0.6584	0.0208
(A,C,M)	(A,M)	**0.4517**	0.4147	0.7929	0.0125
(A,R,C)	(A,R)	0.4092	**0.4214**	0.2354	0.0561
(A,C)	(A)	0.4010	**0.4031**	0.7706	0.0137
(R,C,M)	(R,M)	**0.4013**	0.3488	0.0637	0.0849
(C,M)	(M)	**0.3971**	0.3838	$p < 0.01$	0.1567
(R,C)	(R)	**0.3443**	0.3151	0.0104	0.1162

Table 5.5: Wilcoxon test results for context inclusion and exclusion configurations of the DIT task for all subject systems (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,R,M)	**0.3784**	0.3646	$p < 0.01$	0.2445
(A,C,M)	(A,M)	**0.3785**	0.3537	$p < 0.01$	0.2631
(A,R,C)	(A,R)	**0.3770**	0.3462	$p < 0.01$	0.3748
(A,C)	(A)	**0.3802**	0.3380	$p < 0.01$	0.3302
(R,C,M)	(R,M)	**0.3391**	0.3308	$p < 0.01$	0.2567
(C,M)	(M)	**0.4165**	0.3359	$p < 0.01$	0.3783
(R,C)	(R)	**0.3373**	0.3147	$p < 0.01$	0.4001

and each MRR spread was low. Indeed, all MRR spreads were close, with the largest difference being 0.0525 between (R,C,M) and (R,M). This suggests that for the FLT task, context inclusion is neither notably beneficial nor detrimental. I note that, however, the optimal configuration for the FLT, (A,C,M), includes context, so I cannot advocate for its exclusion.

For the DIT task, however, I get very different results than for FLT, as shown in Table 5.5. All seven pairs improve when context is included, and all seven are statistically significant with somewhat moderate (0.2445) to moderate (0.4001) effect sizes. I also note that the context

Table 5.6: Wilcoxon test results for message inclusion and exclusion configurations of the FLT task for all subject systems (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,R,C)	**0.4315**	0.4092	0.6327	0.0226
(A,C,M)	(A,C)	**0.4517**	0.4010	0.0161	0.1139
(A,R,M)	(A,R)	0.4119	**0.4214**	0.1471	0.0674
(A,M)	(A)	**0.4147**	0.4031	0.0109	0.1195
(R,C,M)	(R,C)	**0.4013**	0.3443	0.0310	0.0984
(C,M)	(C)	**0.3971**	0.3386	0.2897	0.0485
(R,M)	(R)	**0.3488**	0.3151	0.0387	0.0936

text source is included in the optimal configuration for DIT. Together, this suggests that context is worthwhile for inclusion, which does not align with the intuitive view for this source.

5.3.2.3 Messages

I see improvements in the FLT task MRR for nearly all message-including configurations, except for one configuration. As seen in Table 5.6, the configuration (A,R,M) performs worse than configuration (A,R) for FLT. None of the configurations, however, were statistically significant. Thus, like the context source, the inclusion or exclusion of messages is inconclusive under the FLT task and the message source is included in the optimal FLT configuration (A,C,M).

Under the DIT task, Table 5.7, I again see only one configuration that does not favor the inclusion of messages: (A,C,M) compared to (A,C). However, two of the six in favor of inclusion were statistically significant, although each has low effect sizes. Again, the optimal DIT configuration, (C,M), contain the message text source. This aligns with the intuitive view that messages would benefit the model and suggests they should be included.

Table 5.7: Wilcoxon test results for message inclusion and exclusion configurations of the DIT task for all subject systems (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,R,C)	**0.3784**	0.3770	0.0169	0.0995
(A,C,M)	(A,C)	0.3785	**0.3802**	0.1404	0.0616
(A,R,M)	(A,R)	**0.3646**	0.3462	$p < 0.01$	0.1454
(A,M)	(A)	**0.3537**	0.3380	0.5511	0.0241
(R,C,M)	(R,C)	**0.3391**	0.3373	0.3305	0.0399
(C,M)	(C)	**0.4165**	0.4148	0.6168	0.0204
(R,M)	(R)	**0.3308**	0.3147	$p < 0.01$	0.1878

Table 5.8: Wilcoxon test results for removals inclusion and exclusion configurations of the FLT task for all subject systems (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,C,M)	0.4315	**0.4517**	0.2007	0.0620
(A,R,C)	(A,C)	**0.4092**	0.4010	0.4550	0.0356
(A,R,M)	(A,M)	0.4119	**0.4147**	0.0252	0.1053
(A,R)	(A)	**0.4214**	0.4031	0.2649	0.0528
(R,C,M)	(C,M)	**0.4013**	0.3971	0.3120	0.0467
(R,C)	(C)	**0.3443**	0.3386	0.1753	0.0612
(R,M)	(M)	0.3488	**0.3838**	$p < 0.01$	0.2400

Table 5.9: Wilcoxon test results for removals inclusion and exclusion configurations of the DIT task for all subject systems (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,C,M)	0.3784	**0.3785**	$p < 0.01$	0.1405
(A,R,C)	(A,C)	0.3770	**0.3802**	$p < 0.01$	0.1162
(A,R,M)	(A,M)	**0.3646**	0.3537	$p < 0.01$	0.1847
(A,R)	(A)	**0.3462**	0.3380	0.0596	0.0779
(R,C,M)	(C,M)	0.3391	**0.4165**	$p < 0.01$	0.4621
(R,C)	(C)	0.3373	**0.4148**	$p < 0.01$	0.3971
(R,M)	(M)	0.3308	**0.3359**	$p < 0.01$	0.1625

5.3.2.4 Removals

Table 5.8 shows the Wilcoxon test results of removal inclusion and exclusion for FLT. I see that there is only one statistically significant result, and that it favors the intuitive view of excluding removals. Only two of the other six results also favor exclusion. However, it is notable that removals are not included in the optimal FLT configuration. This suggests that, under the FLT, it is largely inconclusive whether removals should be included or excluded.

I see many more significant results for the DIT task, as shown in Table 5.9. Including removals had a negative impact for five of the configuration pairs, all of which were significant with low (0.1162) to moderate (0.4621) effect sizes. Two of the configuration comparisons favored inclusion of removals, with one being statistically significant with a somewhat low effect size of 0.1847. As in the FLT task, again removals do not make an appearance in the optimal configuration. This suggests that removals should be used with caution, and also aligns with the intuitive view that I would expect them to be harmful.

5.4 Findings

The primary goal of this work is to evaluate the performance and reliability of topic models built on the source code histories, i.e., changeset-based topic modeling, in order to move towards more robust approaches that can be utilized by practitioners.

5.4.1 Feature Location

First, with respect to *Research Problem 1* (RP1), applying my approach to feature location, I ask:

RQ 1.1 Is a changeset-based FLT as accurate as a snapshot-based FLT?

RQ 1.2 Does the accuracy of a changeset-based FLT fluctuate as a project evolves?

I have shown that it is not only possible to use a changeset-based TM for FLT, but also desirable. For *RQ 1.1*, I find that the changeset-based approach can produce a more accurate model for searching over source code elements over the traditional snapshot-based approach. I also find evidence under *RQ 1.2* that batch-evaluated approaches may be both over-informed and under-informed, e.g., models trained on data from the future, but does not make good use data available at particular points of interest.

I have also explored class- and method-level granularity searches with different subject systems in prior work [Corley, Kashuda, and Kraft, 2015]. The results of the study presented here differ slightly, but it is difficult to compare the two, given the data granularities are different. In Corley et al. [2015], I were able to arrive at the same conclusion as *RQ 1.1*, but unable to for *RQ 1.2*. That is, while I found that the changeset-based approach was as accurate as snapshots

(RQ 1.1), I were unable to find that the accuracy of the changeset-based model under historical simulation is consistent with the batch counterpart (RQ 1.2). It is difficult to determine the source of this inconsistency, as the prior work used datasets constructed by other researchers [Moreno et al., 2014]. However, the inconsistency increases suspicion that batch evaluations are inadequate or non-representational of reality. I leave extending the current study and dataset to include class- and method-level granularity searches as future work.

I arrive at the conclusion that the current approaches using batched evaluation do not accurately reflect the model performance or the state of the indexed corpus over time. Both of these issues combined present serious threats to validity of prior work in this field. However, researchers can mitigate both of these threats by using a historical simulation of their approaches, instead of only using a batch evaluation. That does not mean researchers must use online LDA as I have, but pay closer attention to whether the approach considers time in their assumptions. That is, researchers need to be to ensure their search engines are constructed only from data that existed *before* the query they are evaluating.

5.4.2 Developer Identification

For *Research Problem 2* (RP2), applying the approach to developer identification, I ask:

RQ 2.1 Is a changeset-based DIT as accurate as a snapshot-based DIT?

RQ 2.2 Does the accuracy of a changeset-based DIT fluctuate as a project evolves?

In this work I have shown that although it is possible to use a changeset-based TM for DIT, it may not be desirable. For *RQ 2.1*, I find that the changeset-based approach does not produce

a more accurate model for searching over developer profiles over the traditional snapshot-based approach. I also find evidence, however, under *RQ 2.2* that batch-evaluated approaches may be both over-informed and under-informed, e.g., models trained on data from the future, but does not make good use data available at particular points of interest. As in the previous section on my FLT, I arrive at the conclusion that the current approaches using batched evaluation do not accurately reflect the model performance or the state of the indexed corpus over time.

5.4.3 Combining and Configuring Changeset-based Topic Models

I determine for *Research Problem 3* (RP3) whether I can gain model re-use for both tasks by asking the following:

RQ 3.1 Can we use the same topic model in more than one context effectively?

RQ 3.2 What are the effects of using different portions of a changeset for corpus construction, such as added, removed, context lines, and the commit message?

There will always be improvement by hyper-optimizing the configuration of a topic model and corpus construction. The results for *RQ 3.1* suggest that it is possible reuse the same model for two tasks, as differences are often by one configuration parameter and not statistically significant. Corpus construction, however, would appear to require more consideration towards which task is more important as there are statistically significant effects between the optimal and alternate configurations.

As shown in results Section 4.3, there are effects of choosing differing text sources. I have discussed each of these text sources, comparing configurations that include a particular source to the

equivalent configuration that excludes that same source. The results and discussion surrounding *RQ 3.2* suggest that it is beneficial to include additions, context, and messages while excluding removals. This tends to match the intuitive view that removals would be detrimental to the performance of the FLT and DIT.

5.5 Threats to Validity

My studies have limitations that impact the validity of my findings, as well as my ability to generalize them. I describe some of these limitations and their impacts.

5.5.1 Construct Validity

Threats to construct validity concern measurements accurately reflecting the features of interest. A possible threat to construct validity is the benchmarks. Errors in the datasets could result in inaccurate effectiveness measures. The dataset creation technique closely follows that of other researchers [Dit et al., 2013; Moreno et al., 2014; Revelle, Dit, and Poshyvanyk, 2010]. Additionally, datasets extracted source code entities automatically from changesets, previous work shows this approach is on par with manual extraction [Corley, Kraft, Etzkorn, and Lukins, 2011].

5.5.2 Internal Validity

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

Another threat to internal validity pertains to the value of parameters such as K that I selected

for all models trained. I decided that the snapshot- and changeset-based approaches should have the same parameters to help facilitate evaluation and comparison. I argue that the study is not about selecting the best parameters, but to show that the snapshot-based approach is reasonable.

Further, since LDA implementations such as Gensim rely heavily on randomly initialized matrices, I have determined a certain threat with respect to this model initialization. I 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 of 1 on the system's pseudo-random number generator.

5.5.3 External Validity

Threats to external validity concern the extent to which I can generalize the results. The subjects of the study comprise six open source projects in Java, so I cannot generalize the 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.

5.5.4 Conclusion Validity

Threats to conclusion validity concern the choice of measurements and how those choices impact the evaluation and conclusion. I chose to use mean reciprocal rank (MRR), but I could have also used mean average precision (MAP) instead. I chose the former because it lends itself to being paired with the Wilcoxon signed-rank test as both rely on the same input data. I also chose the former to establish a baseline that can be used to compare my approach against prior work in the field [Dit et al., 2013].

6. CONCLUSION

This work explores a changeset-based approach and evaluates the approach against the state-of-the-art snapshot-based topic model search engine in two contexts: feature location and developer identification. I investigate the performance of a changeset-based model in a historical simulation, wherein I see how a stream of changes would impact issue queries closer to real time. Finally, I discuss an approach for using a singular topic model for both of these tasks and consider configuration needs of such a multi-tasked model.

For *Research Problem 1* (RP1), feature location, I addressed two research questions regarding the performance of a TM-based FLT trained on changesets. First, I compare a batch TM-based FLT trained on the changesets of a project's history to one trained on the snapshot of source code entities. I found that changesets can perform as well as or better than snapshots. Second, I compare a batch TM-based FLT trained on changesets to a historical simulation of a TM-based FLT trained on the same changesets over time. I show that the historical simulation more accurately portrays how a FLT would execute in a real environment. This study extended Corley et al. [2015] to the file-level granularity, where I found similar results for methods and classes.

I found for developer identification, in *Research Problem 2* (RP2), a different conclusion from RP1. Again, I compare a batch TM-based DIT trained on the changesets of a project's history to one trained on the snapshot of source code entities and find the former does not perform as well as the latter. Next, I compare a batch TM-based DIT trained on changesets to a historical simulation

of a TM-based DIT trained on the same changesets over time. I find that the historical simulation may more accurately portray how a DIT would execute in a real environment. I also see under historical simulation results that show less fluctuation for DIT than FLT, though some fluctuation does exist.

Finally, in *Research Problem 3* (RP3), I explore whether using the same changeset-based model for two different tasks is feasible and what configuration considerations might need to be taken to construct a successful model. I found that the same model can be reused for two tasks, so long as trade-offs can be made. It will always be possible to optimize parameters and inputs for a particular task and subject system, but the study suggests it is possible to have acceptable performance across tasks. However, I found there were significant effects in corpus construction, which will require more careful consideration when choosing to optimize for a task.

My personal perspective from industry is as follows. First, search tools, i.e., feature location tools, aren't used by developers because they often aren't needed, even in large systems. However, the caveat to this is that the system I've encountered are generally modular and abstracted properly enough to mitigate this issue. I would like to see more work towards feature location as applied to poorly engineered systems, which may face new problems. Rather, I see feature location tools being more useful for feature discovery for external users and stakeholders, who are often not aware of internal system implementation details or progress.

I do believe there is much more usefulness in a developer identification tool. Anecdotally, I often have the need to find other developers to gain perspective as to *why* a system behaves a certain way. Additionally, often it seems that external users and stakeholders need to be able to quickly find a developer to gain similar perspectives. One issue with the approach many DITs take is that

they ultimately rely on an FLT or variant thereof. Hence, while I see the most impact in industry with improving DITs, there will also be a need to continue to work towards improving FLTs and the insights gained will impact both.

The results encourage the idea that there is still much to explore in the area of feature location and developer identification. What other untapped resources might be available? I show changesets are yet another viable resource researchers and practitioners should be taking advantage of for the feature location task. For example, future work includes investigating whether, along with the message text source, code reviews are valuable. Likewise, it may also be beneficial to begin including changes to other systems the subject system is dependent on, as call-site inclusion may benefit the model [Bassett and Kraft, 2013].

The results also show that research remains not only in improving accuracies of FLTs and DITs, but also in solving the practical aspects of building FLTs and DITs that are robust *and* agile enough to keep up with fast-changing software. This includes work not just with topic models, such as LDA, but promising deep learning models, such as Doc2Vec [Corley, Damevski, and Kraft, 2015]. I hope future work with deep learning models includes extension of the historical simulation work once more of these deep learning algorithms can be updated online, instead of in batch.

Additional future work exists in regard to configuration. In a changeset it may be desirable to parse further for source code entities using island grammar parsing [Moonen, 2001]. As shown, it is desirable to only use portions of the changeset, hence extracting changes between the abstract syntax trees [Fluri, Wursch, and Gall, 2007] may be just as fruitful for gaining more precise changesets. The experiments explored in this work were non-exhaustive. For example, the corpus construction sweep for DIT only included a sweep of the model corpus, i.e., the source changesets, but not

of the indexed corpus, i.e. the developer profile corpus, which was only the default configuration of (A,R,C). Most importantly, like previous work [Biggers et al., 2014], it would be wise to further investigate the two online LDA variables, τ_0 and κ, to measure their effect on the historical simulation.

I also would like to expand the historical simulation parts of this study to include both snapshots and changesets, as it would be useful to compare results between batch snapshots and simulated snapshots. I did not conduct a historical evaluation of the snapshot approaches due to the brute-force nature of the approach increasing the amount of time the experiment would take. In this future work, a new model would be constructed for each issue at the time of the query, rather than a single model constructed at the end of a release cycle. Performing a historical snapshot experiment would also likely decrease the number of failures that occur during query-time, as the model and index would be more representative of the code that that time. However, there has yet to be an in-depth study, to my best knowledge, of why failures occur in the snapshot approach.

I found that I had 12 query failures during the DIT historical simulation. For DIT, this was due to that commit being the author's first check-in under that alias. The non-historical simulations succeeded on each of these 12 failures since they had knowledge of every developer to have worked on that project. Indeed, better alias linking could potentially alleviate some of this issue, but would not remedy it entirely. Future work towards resolving these failures could include integrating "insider knowledge" about when new members join a team or project. This sort of insight could also boost DIT performance once members that have left a team or project are no longer being recommended.

Finally, a closer look at the experimental approach used in modern FLT and DIT is needed,

in particular how models vary with different random starting states. I have observed it possible to change the result of an experiment by simply re-running the experiment without setting a random seed, i.e., without setting a random seed before an experiment the results are not reproducible. Topic models, like LDA, are known to get "stuck" in local minima [Binkley, Heinz, Lawrie, and Overfelt, 2014] due to its non-convexity. Evaluating and comparing techniques based on single runs does not show whether one is somehow better or more effective than another. In order to accurately judge an approach in comparison to another, I must run multiple experiments to show that one approach typically scores higher than another. This allows for me to eliminate outlier results in order to create a more fair comparison.

REFERENCES

Abadi, A., M. Nisenson, and Y. Simionovici (2008). A traceability technique for specifications. In *Proceedings of the 16th International Conference on Program Comprehension*, pp. 103–112. IEEE.

Agrawal, A., W. Fu, and T. Menzies (2018). What is wrong with topic modeling? And how to fix it using search-based software engineering. *Information and Software Technology*, 1–15.

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

Anvik, J., L. Hiew, and G. C. Murphy (2006). Who should fix this bug? In *Proceedings of the 28th International Conference on Software Engineering*, pp. 361–370. ACM.

Anvik, J. and G. C. Murphy (2007). Determining implementation expertise from bug reports. In *Proceedings of the 4th International Workshop on Mining Software Repositories*, pp. 2–10. IEEE.

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

Baldi, P. F., C. V. Lopes, E. J. Linstead, and S. K. Bajracharya (2008). A theory of aspects as latent topics. In *Proceedings of the 23rd ACM SIGPLAN Conference on Object-oriented Programming Systems Languages and Applications*, pp. 543–562. ACM.

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

Begel, A., Y. P. Khoo, and T. Zimmermann (2010). Codebook: discovering and exploiting relationships in software repositories. In *Proceedings of the 32nd International Conference on Software Engineering*, Volume 1, pp. 125–134. IEEE.

Bhattacharya, P., I. Neamtiu, and C. R. Shelton (2012). Automated, highly-accurate, bug assignment using machine learning and tossing graphs. *Journal of Systems and Software 85*(10), 2275–2292.

Biggers, L. R. (2012). *Investigating the effect of corpus construction on latent Dirichlet allocation based feature location*. Ph. D. thesis, University of Alabama.

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

Binkley, D., D. Heinz, D. Lawrie, and J. Overfelt (2014). Understanding LDA in source code analysis. In *Proceedings of the International Conference on Program Comprehension*, pp. 26–36. ACM.

Bird, C., N. Nagappan, B. Murphy, H. Gall, and P. Devanbu (2011). 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*, pp. 4–14. ACM.

Bissyande, T. F., F. Thung, S. Wang, D. Lo, L. Jiang, and L. Reveillere (2013). Empirical evaluation of bug linking. In *Proceedings of the 17th European Conference on Software Maintenance and Reengineering*, pp. 89–98. IEEE.

Blei, D. (2012). Probabilistic topic models. *Communications of the ACM 55*(4), 77–84.

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

Bortis, G. and A. Van der Hoek (2013). PorchLight: a tag-based approach to bug triaging. In *Proceedings of the 35th International Conference on Software Engineering*, pp. 342–351. IEEE.

Brand, M. (2006). Fast low-rank modifications of the thin singular value decomposition. *Linear algebra and its applications 415*(1), 20–30.

Canfora, G. and L. Cerulo (2006). Supporting change request assignment in open source development. In *Proceedings of the 2006 ACM Symposium on Applied Computing*, pp. 1767–1772. ACM.

Cataldo, M., P. A. Wagstrom, J. D. Herbsleb, and K. M. Carley (2006). Identification of coordination requirements: implications for the design of collaboration and awareness tools. In *Proceedings of the 20th Anniversary Conference on Computer Supported Cooperative Work*, pp. 353–362. ACM.

Chang, J. and D. M. Blei (2010). Hierarchical relational models for document networks. *The Annals of Applied Statistics 4*(1), 124–150.

Chaparro, O., J. M. Florez, and A. Marcus (2017). Using observed behavior to reformulate

queries during text retrieval-based bug localization. In *Proceedings of the International Conference on Software Maintenance and Evolution*. IEEE.

Corbi, T. A. (1989). Program understanding: challenge for the 1990s. *IBM Systems Journal 28*(2), 294–306.

Corley, C. S., K. Damevski, and N. A. Kraft (2015). Exploring the use of deep learning for feature location. In *Proceedings of the International Conference on Software Maintenance and Evolution*, pp. 556–560. IEEE.

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

Corley, C. S., K. L. Kashuda, and N. A. Kraft (2015). Modeling changeset topics for feature location. In *Proceedings of the International Conference on Software Maintenance and Evolution*, pp. 71–80. IEEE.

Corley, C. S., K. L. Kashuda, D. S. May, and N. A. Kraft (2014). Modeling changeset topics. In *Proceedings of the 4th Workshop on Mining Unstructured Data*, pp. 6–10. IEEE.

Corley, C. S., N. A. Kraft, L. H. Etzkorn, and S. K. Lukins (2011). Recovering traceability links between source code and fixed bugs via patch analysis. In *Proceedings of the 6th International Workshop on Traceability in Emerging Forms of Software Engineering*, pp. 31–37. ACM.

Croft, B., D. Metzler, and T. Strohman (2010). *Search Engines: Information Retrieval in Practice*. Addison-Wesley.

Cubranic, D. and G. Murphy (2004). Automatic bug triage using text categorization. In *Proceedings of the 16th International Conference on Software Engineering & Knowledge Engineering*, pp. 92–97. Citeseer.

Cubranic, D., G. C. Murphy, J. Singer, and K. S. Booth (2005). Hipikat: a project memory for software development. *IEEE Transactions on Software Engineering 31*(6), 446–465.

Deerwester, S. C., S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman (1990). Indexing by latent semantic analysis. *Journal of the American Society of Information Science 41*(6), 391–407.

Dit, B., L. Guerrouj, D. Poshyvanyk, and G. Antoniol (2011). Can better identifier splitting techniques help feature location? In *Proceedings of the 19th International Conference on Program Comprehension*. IEEE.

Dit, B., A. Holtzhauer, D. Poshyvanyk, and H. Kagdi (2013). A dataset from change history

to support evaluation of software maintenance tasks. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pp. 131–134. IEEE.

Dit, B., M. Revelle, M. Gethers, and D. Poshyvanyk (2013). Feature location in source code: a taxonomy and survey. *Journal of Software: Evolution and Process 25*(1), 53–95.

Dit, B., M. Revelle, and D. Poshyvanyk (2012). Integrating information retrieval, execution and link analysis algorithms to improve feature location in software. *Empirical Software Engineering 18*(2), 277–309.

Eaddy, M., A. V. Aho, G. Antoniol, and Y.-G. Guéhéneuc (2008). Cerberus: tracing requirements to source code using information retrieval, dynamic analysis, and program analysis. In *Proceedings of the 16th International Conference on Program Comprehension*, pp. 53–62. IEEE.

Eddy, B. P., N. A. Kraft, and J. Gray (2017). Impact of structural weighting on a latent Dirichlet allocation-based feature location technique. *Journal of Software: Evolution and Process 30*(1), 1–25.

Fluri, B., M. Wursch, and H. C. Gall (2007). Do code and comments co-evolve? On the relation between source code and comment changes. In *Proceedings of the 14th Working Conference on Reverse Engineering*, pp. 70–79. IEEE.

Fox, C. (1992). Lexical analysis and stoplists. In W. B. Frakes and R. Baeza-Yates (Eds.), *Information Retrieval: Data Structures and Algorithms*, pp. 102–130. Prentice-Hall.

Fritz, T., G. C. Murphy, and E. Hill (2007). Does a programmer's activity indicate knowledge of code? In *Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pp. 341–350. ACM.

Gethers, M., T. Savage, M. Di Penta, R. Oliveto, D. Poshyvanyk, and A. De Lucia (2011). CodeTopics: which topic am I coding now? In *Proceedings of the 33rd International Conference on Software Engineering*, pp. 1034–1036. ACM.

Gorrell, G. and B. Webb (2005). Generalized Hebbian algorithm for incremental latent semantic analysis. In *Proceedings of INTERSPEECH*, pp. 1325–1328. Citeseer.

Guo, P. J., T. Zimmermann, N. Nagappan, and B. Murphy (2010). Characterizing and predicting which bugs get fixed: an empirical study of Microsoft Windows. In *Proceedings of the 32nd International Conference on Software Engineering*, pp. 495–504. ACM.

Guo, P. J., T. Zimmermann, N. Nagappan, and B. Murphy (2011). "Not my bug!" and other reasons for software bug report reassignments. In *Proceedings of the Conference on Computer Supported Cooperative Work*, pp. 395–404. ACM.

Halko, N., P.-G. Martinsson, and J. A. Tropp (2011). Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review 53*(2), 217–288.

Herbsleb, J. D., A. Mockus, T. A. Finholt, and R. E. Grinter (2001). An empirical study of global software development: distance and speed. In *Proceedings of the 23rd International Conference on Software Engineering*, pp. 81–90. IEEE.

Hoffman, M., F. R. Bach, and D. M. Blei (2010). Online learning for latent Dirichlet allocation. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta (Eds.), *Advances in Neural Information Processing Systems 23*, pp. 856–864. Curran Associates, Inc.

Hossen, M. K., H. Kagdi, and D. Poshyvanyk (2014). Amalgamating source code authors, maintainers, and change proneness to triage change requests. In *Proceedings of the 22nd International Conference on Program Comprehension*. ACM.

Järvelin, K. and J. Kekäläinen (2002). Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems 20*(4), 422–446.

Jeong, G., S. Kim, and T. Zimmermann (2009). 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*, pp. 111–120. ACM.

Just, R., D. Jalali, and M. D. Ernst (2014). Defects4j: a database of existing faults to enable controlled testing studies for java programs. In *Proceedings of the International Symposium on Software Testing and Analysis*, pp. 437–440. ACM.

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

Kagdi, H., M. Hammad, and J. I. Maletic (2008). Who can help me with this source code change? In *Proceedings of the International Conference on Software Maintenance*, pp. 157–166. IEEE.

Kerby, D. S. (2014). The simple difference formula: an approach to teaching nonparametric correlation. *Innovative Teaching 3*(1), 1–9.

Ko, A. J., B. A. Myers, M. J. Coblenz, and H. H. Aung (2006). An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. *IEEE Transactions on Software Engineering 32*(12), 971–987.

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

Levey, A. and M. Lindenbaum (2000). Sequential Karhunen-Loeve basis extraction and its application to images. *IEEE Transactions on Image Processing 9*(8), 1371–1374.

Linares-Vásquez, M., K. Hossen, H. Dang, H. Kagdi, M. Gethers, and D. Poshyvanyk (2012). Triaging incoming change requests: bug or commit history, or code authorship? In *Proceedings of the 28th International Conference on Software Maintenance*, pp. 451–460. IEEE.

Linstead, E., P. Rigor, S. Bajracharya, C. Lopes, and P. Baldi (2007). Mining eclipse developer contributions via author-topic models. In *Proceedings of the 4th Working Conference on Mining Software Repositories*, pp. 30–30. IEEE.

Liu, D., A. Marcus, D. Poshyvanyk, and V. Rajlich (2007). 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*, pp. 234–243. ACM.

Liu, J., Y. Tian, X. Yu, Z. Yang, X. Jia, C. Ma, and Z. Xu (2016). A multi-source approach for bug triage. *International Journal of Software Engineering and Knowledge Engineering 26*(09n10), 1593–1604.

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

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

Ma, D., D. Schuler, T. Zimmermann, and J. Sillito (2009). Expert recommendation with usage expertise. In *Proceedings of the International Conference on Software Maintenance*, Volume 0, pp. 535–538. IEEE.

Madsen, R. E., S. Sigurdsson, L. K. Hansen, and J. Larsen (2004). Pruning the vocabulary for better context recognition. In *Proceedings of the 17th International Conference on Pattern Recognition*, Volume 2, pp. 483–488. IEEE.

Manning, C. D., P. Raghavan, and H. Schütze (2008). *Introduction to Information Retrieval*, Volume 1. Cambridge University Press.

Marcus, A. and T. Menzies (2010). Software is data too. In *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, pp. 229–232. ACM.

Marcus, A. and D. Poshyvanyk (2005). The conceptual cohesion of classes. In *Proceedings of the 21st International Conference on Software Maintenance*, pp. 133–142.

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

Maskeri, G., S. Sarkar, and K. Heafield (2008). Mining business topics in source code using latent Dirichlet allocation. In *Proceedings of the 1st India Software Engineering Conference*, pp. 113–120. ACM.

Matter, D., A. Kuhn, and O. Nierstrasz (2009). Assigning bug reports using a vocabulary-based expertise model of developers. In *Proceedings of the 6th International Working Conference on Mining Software Repositories*, pp. 131–140. IEEE.

McDonald, D. W. and M. S. Ackerman (1998). Just talk to me: a field study of expertise location. In *Proceedings of the Conference on Computer Supported Cooperative Work*, pp. 315–324. ACM.

McDonald, D. W. and M. S. Ackerman (2000). Expertise recommender: a flexible recommendation system and architecture. In *Proceedings of the Conference on Computer Supported Cooperative Work*, pp. 231–240. ACM.

Mills, C., G. Bavota, S. Haiduc, R. Oliveto, A. Marcus, and A. D. Lucia (2017). Predicting query quality for applications of text retrieval to software engineering tasks. *ACM Transactions on Software Engineering and Methodology 26*(1), 1–45.

Minto, S. and G. C. Murphy (2007). Recommending emergent teams. In *Proceedings of the 4th International Workshop on Mining Software Repositories*, pp. 5. IEEE.

Mockus, A. and J. D. Herbsleb (2002). Expertise browser: a quantitative approach to identifying expertise. In *Proceedings of the 24th International Conference on Software Engineering*, pp. 503–512. ACM.

Moonen, L. (2001). Generating robust parsers using island grammars. In *Proceedings of the 8th Working Conference on Reverse Engineering*, pp. 13–22. IEEE.

Moreno, L., J. J. Treadway, A. Marcus, and W. Shen (2014). On the use of stack traces to improve text retrieval-based bug localization. In *Proceedings of the International Conference on Software Maintenance and Evolution*, pp. 151–160. IEEE.

Müller, H. A., J. H. Jahnke, D. B. Smith, M.-A. Storey, S. R. Tilley, and K. Wong (2000). Reverse engineering: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, pp. 47–60. ACM.

Ouni, A., R. G. Kula, and K. Inoue (2016). Search-based peer reviewers recommendation in modern code review. In *Proceedings of the International Conference on Software Maintenance and Evolution*, pp. 367–377. IEEE.

Petrenko, M., V. Rajlich, and R. Vanciu (2008). Partial domain comprehension in software evolution and maintenance. In *Proceedings of the 16th International Conference on Program Comprehension*, pp. 13–22. IEEE.

Porter, M. F. (1980). An algorithm for suffix stripping. *Program: Electronic Library and Information Systems 14*(3), 130–137.

Poshyvanyk, D., Y.-G. Gueheneuc, A. Marcus, G. Antoniol, and V. Rajlich (2006). Combining probabilistic ranking and latent semantic indexing for feature identification. In *Proceedings of the 14th International Conference on Program Comprehension*, pp. 137–148. IEEE.

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

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

Rahman, F. and P. Devanbu (2011). Ownership, experience and defects: a fine-grained study of authorship. In *Proceedings of the 33rd International Conference on Software Engineering*, pp. 491–500. ACM.

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

Rao, S. (2013). *Incremental update framework for efficient retrieval from software libraries for bug localization*. Ph. D. thesis, Purdue University.

Rao, S., H. Medeiros, and A. Kak (2013). An incremental update framework for efficient retrieval from software libraries for bug localization. In *Proceedings of the 20th Working Conference on Reverse Engineering*, pp. 62–71. IEEE.

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

Saha, R. K., J. Lawall, S. Khurshid, and D. E. Perry (2014). On the effectiveness of information retrieval based bug localization for C programs. In *Proceedings of the International Conference on Software Maintenance and Evolution*. IEEE.

Saha, R. K., M. Lease, S. Khurshid, and D. E. Perry (2013). Improving bug localization using structured information retrieval. In *Proceedings of the 28th International Conference on Automated Software Engineering*. IEEE.

Salton, G. and C. Buckley (1988). Term-weighting approaches in automatic text retrieval. *Information Processing and Management 24*(5), 513–523.

Salton, G. and M. J. McGill (1983). *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc.

Salton, G., A. Wong, and C.-S. Yang (1975). A vector space model for automatic indexing. *Communications of the ACM 18*(11), 613–620.

Saul, Z. M., V. Filkov, P. Devanbu, and C. Bird (2007). Recommending random walks. In *Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pp. 15–24. ACM.

Savage, T., B. Dit, M. Gethers, and D. Poshyvanyk (2010). Topic XP: exploring topics in source code using latent Dirichlet allocation. In *Proceedings of the International Conference on Software Maintenance*, pp. 1–6. IEEE.

Scanniello, G. and A. Marcus (2011). Clustering support for static concept location in source code. In *Proceedings of the 19th International Conference on Program Comprehension*, pp. 1–10. IEEE.

Schuler, D. and T. Zimmermann (2008). Mining usage expertise from version archives. In *Proceedings of the 2008 International Working Conference on Mining Software Repositories*, pp. 121–124. ACM.

Shao, P., T. Atkison, N. A. Kraft, and R. K. Smith (2012). Combining lexical and structural information for static bug localisation. *International Journal of Computer Applications in Technology 44*(1), 61.

Shokripour, R., J. Anvik, Z. M. Kasirun, and S. Zamani (2013). Why so complicated? Simple term filtering and weighting for location-based bug report assignment recommendation. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pp. 2–11. IEEE.

Sisman, B. and A. C. Kak (2013). Assisting code search with automatic query reformulation for bug localization. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pp. 309–318. IEEE.

Somasundaram, K. and G. C. Murphy (2012). Automatic categorization of bug reports using

latent Dirichlet allocation. In *Proceedings of the 5th India Software Engineering Conference*, pp. 125–130. ACM.

Steyvers, M., P. Smyth, M. Rosen-Zvi, and T. Griffiths (2004). Probabilistic author-topic models for information discovery. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 306–315. ACM.

Tamrawi, A., T. T. Nguyen, J. M. Al-Kofahi, and T. N. Nguyen (2011). 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*, pp. 365–375. ACM.

Teh, Y. W., M. I. Jordan, M. J. Beal, and D. M. Blei (2006). Hierarchical Dirichlet processes. *Journal of the american statistical association 101*(476), 1566–1581.

Vasa, R., J.-G. Schneider, and O. Nierstrasz (2007). The inevitable stability of software change. In *Proceedings of the International Conference on Software Maintenance*, pp. 4–13. IEEE.

Řehůřek, R. (2011). Subspace tracking for latent semantic analysis. In *Advances in Information Retrieval*, pp. 289–300. Springer.

Řehůřek, R. and P. Sojka (2010). Software framework for topic modelling with large corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pp. 45–50. ELRA.

Wang, S., F. Khomh, and Y. Zou (2013). Improving bug localization using correlations in crash reports. In *Proceedings of the 10th Working Conference on Mining Software Repositories*. IEEE.

Weissgerber, P., M. Pohl, and M. Burch (2007). Visual data mining in software archives to detect how developers work together. In *Proceedings of the 4th International Workshop on Mining Software Repositories*, pp. 9. IEEE.

Wong, C.-P., Y. Xiong, H. Zhang, D. Hao, L. Zhang, and H. Mei (2014). Boosting bug-report-oriented fault localization with segmentation and stack-trace analysis. In *Proceedings of the International Conference on Software Maintenance and Evolution*, pp. 181–190. IEEE.

Ying, A. T. T. and M. P. Robillard (2013). Code fragment summarization. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, pp. 655–658. ACM.

Youm, K. C., J. Ahn, and E. Lee (2017). Improved bug localization based on code change histories and bug reports. *Information and Software Technology 82*(Supplement C), 177–192.

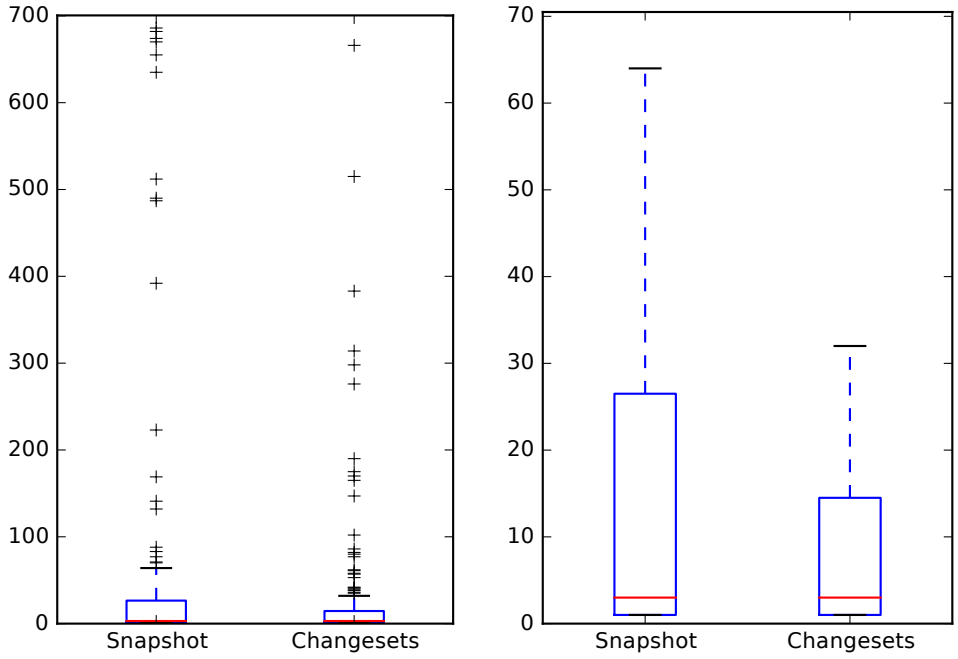
Zha, H. and H. D. Simon (1999). On updating problems in latent semantic indexing. *SIAM Journal on Scientific Computing 21*(2), 782–791.

Zhai, K. and J. Boyd-Graber (2013). Online latent Dirichlet allocation with infinite vocabulary. In *Proceedings of the 30th International Conference on Machine Learning*, pp. 561–569. PMLR.

Zhou, Y., Y. Tong, T. Chen, and J. Han (2017). Augmenting bug localization with part-of-speech and invocation. *International Journal of Software Engineering and Knowledge Engineering 27*(06), 925–949.

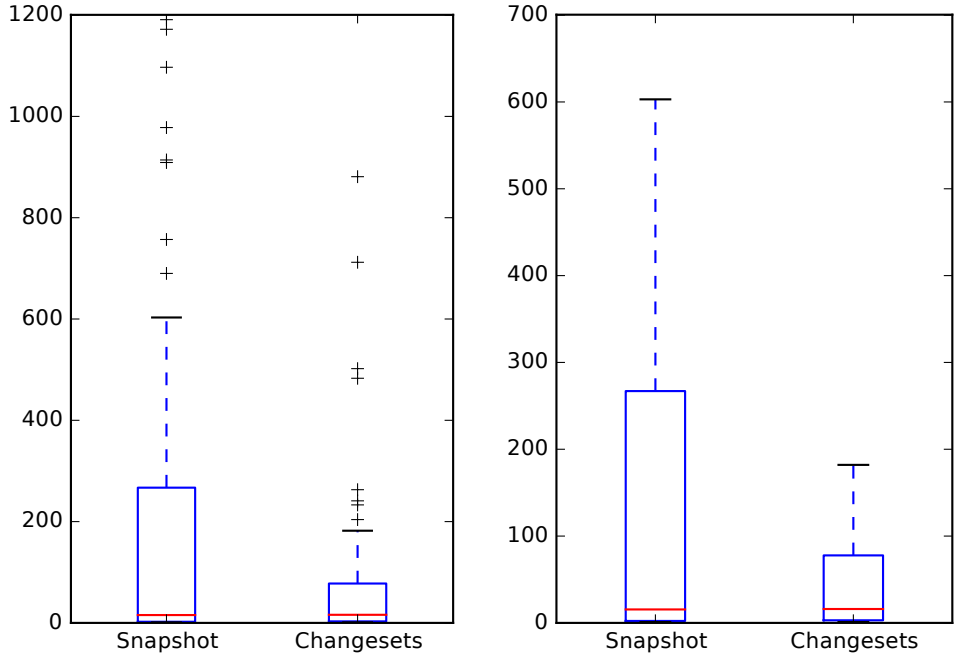
A. RP1: FEATURE LOCATION BOXPLOTS

In these figures, we display the effectiveness measures for each subject system for *Research Problem 1* (RP1).



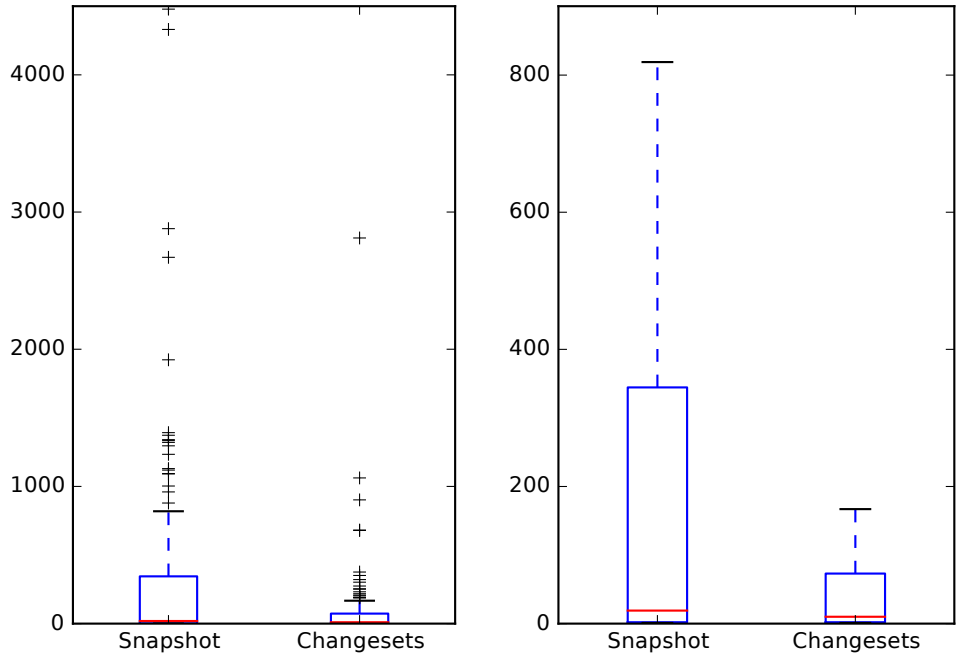
(a) Including outliers (b) Excluding outliers

Figure A.1: *RQ 1.1*: Feature Location effectiveness measures for BookKeeper v4.3.0



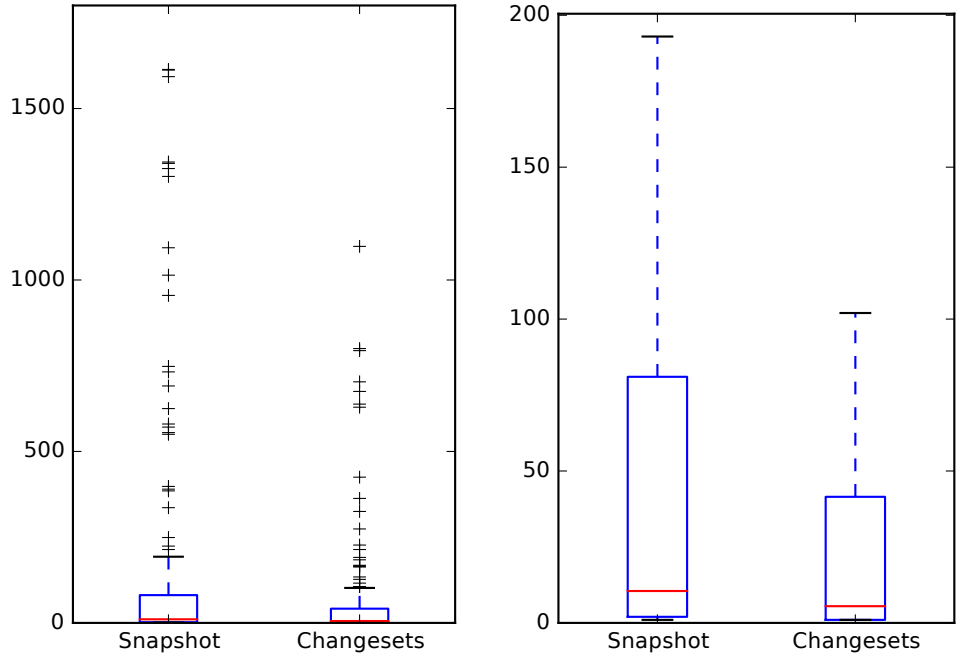
(a) Including outliers (b) Excluding outliers

Figure A.2: *RQ 1.1*: Feature Location effectiveness measures for Mahout v0.10.0



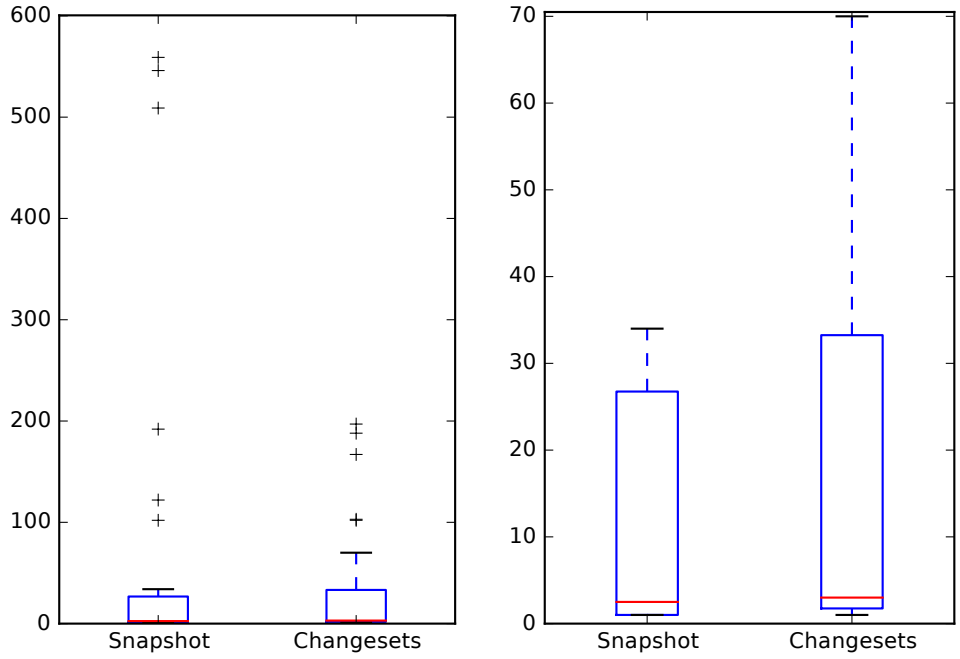
(a) Including outliers (b) Excluding outliers

Figure A.3: *RQ 1.1*: Feature Location effectiveness measures for OpenJPA v2.3.0



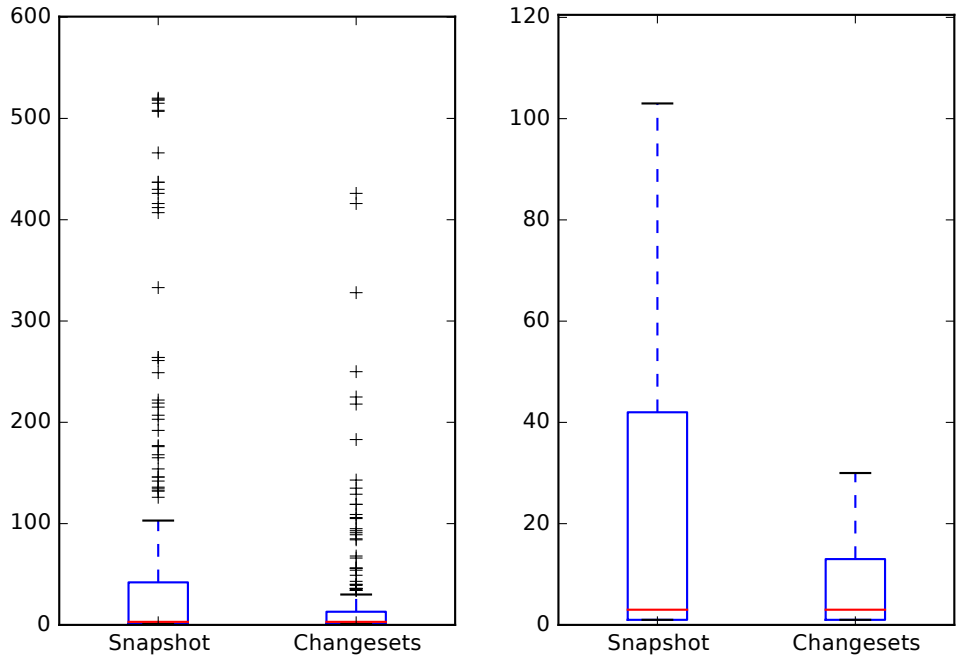
(a) Including outliers (b) Excluding outliers

Figure A.4: *RQ 1.1*: Feature Location effectiveness measures for Pig v0.14.0



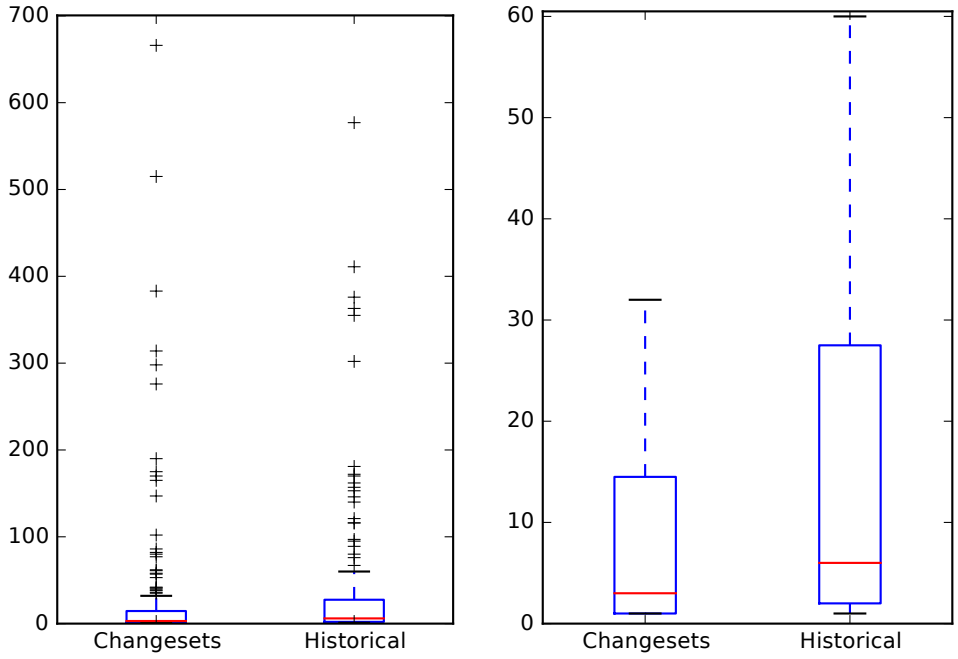
(a) Including outliers (b) Excluding outliers

Figure A.5: *RQ 1.1*: Feature Location effectiveness measures for Tika v1.8



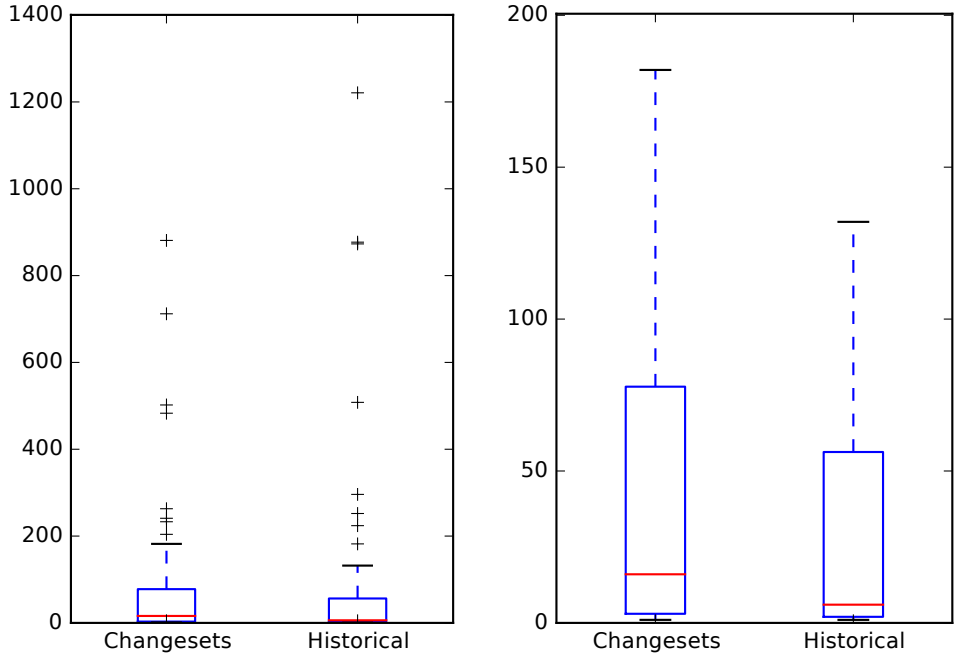
(a) Including outliers (b) Excluding outliers

Figure A.6: *RQ 1.1*: Feature Location effectiveness measures for ZooKeeper v3.5.0



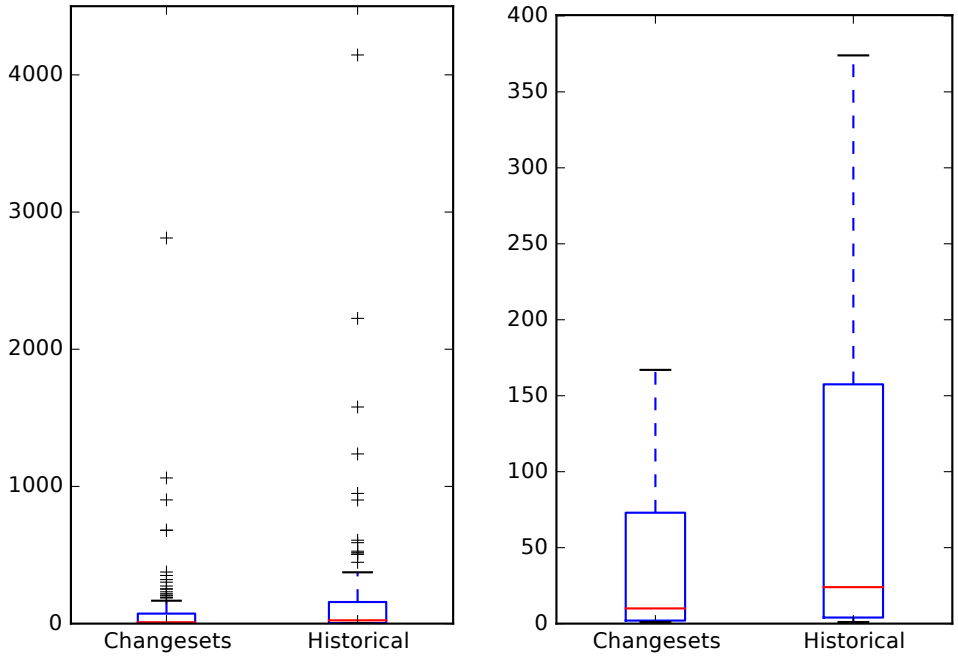
(a) Including outliers (b) Excluding outliers

Figure A.7: *RQ 1.2*: Feature Location effectiveness measures for BookKeeper v4.3.0



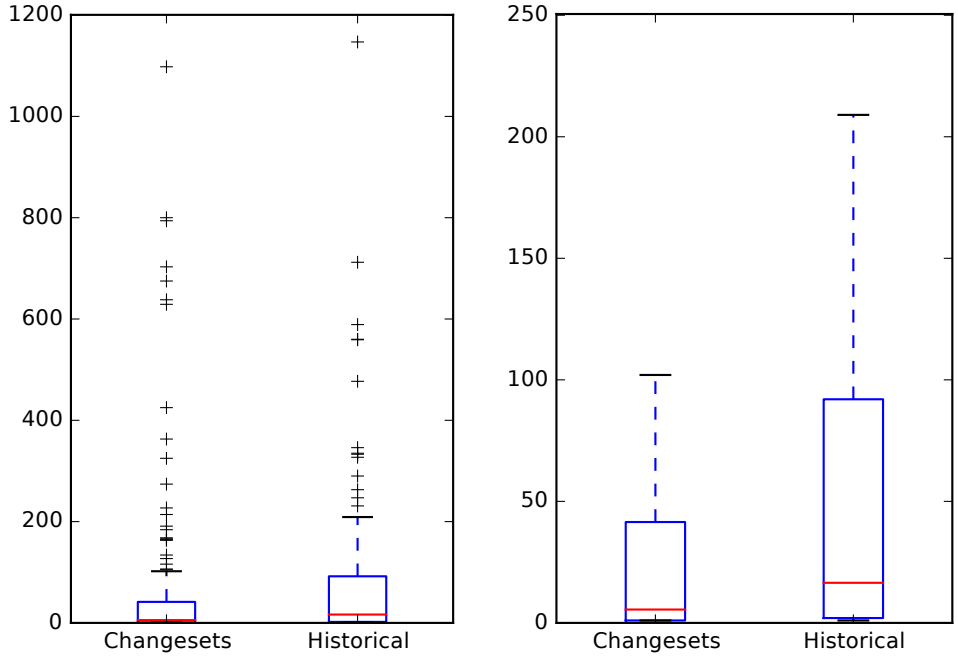
(a) Including outliers (b) Excluding outliers

Figure A.8: *RQ 1.2*: Feature Location effectiveness measures for Mahout v0.10.0



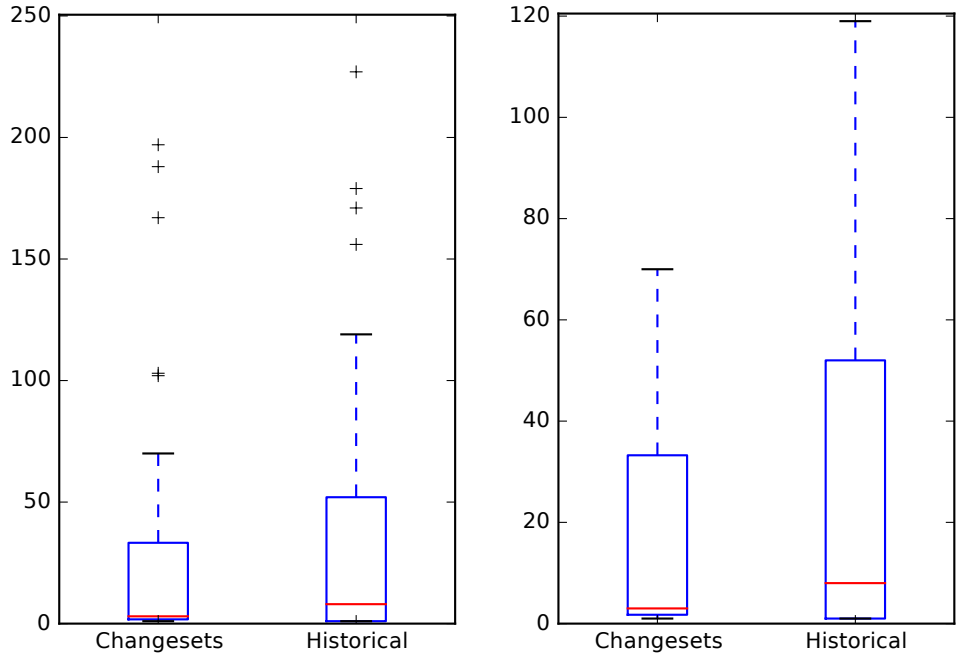
(a) Including outliers (b) Excluding outliers

Figure A.9: *RQ 1.2*: Feature Location effectiveness measures for OpenJPA v2.3.0



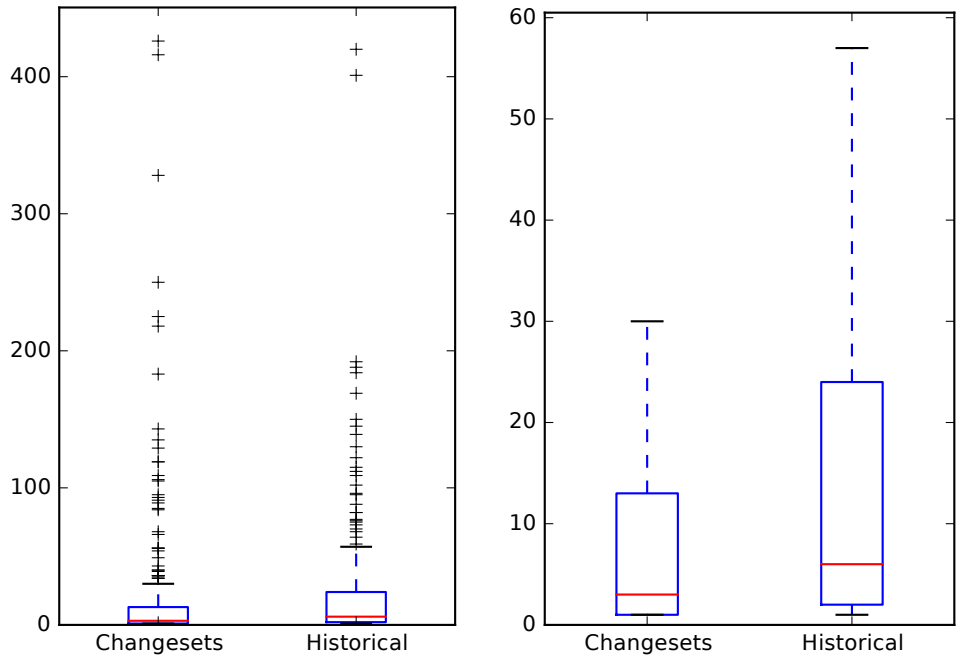
(a) Including outliers (b) Excluding outliers

Figure A.10: *RQ 1.2*: Feature Location effectiveness measures for Pig v0.14.0



(a) Including outliers (b) Excluding outliers

Figure A.11: *RQ 1.2*: Feature Location effectiveness measures for Tika v1.8

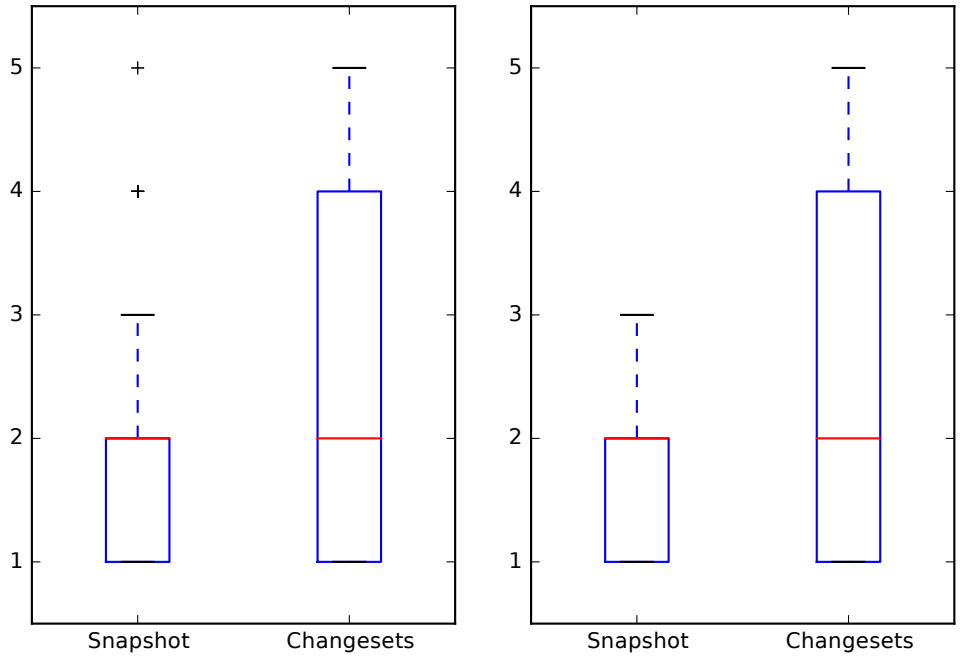


(a) Including outliers (b) Excluding outliers

Figure A.12: *RQ 1.2*: Feature Location effectiveness measures for ZooKeeper v3.5.0

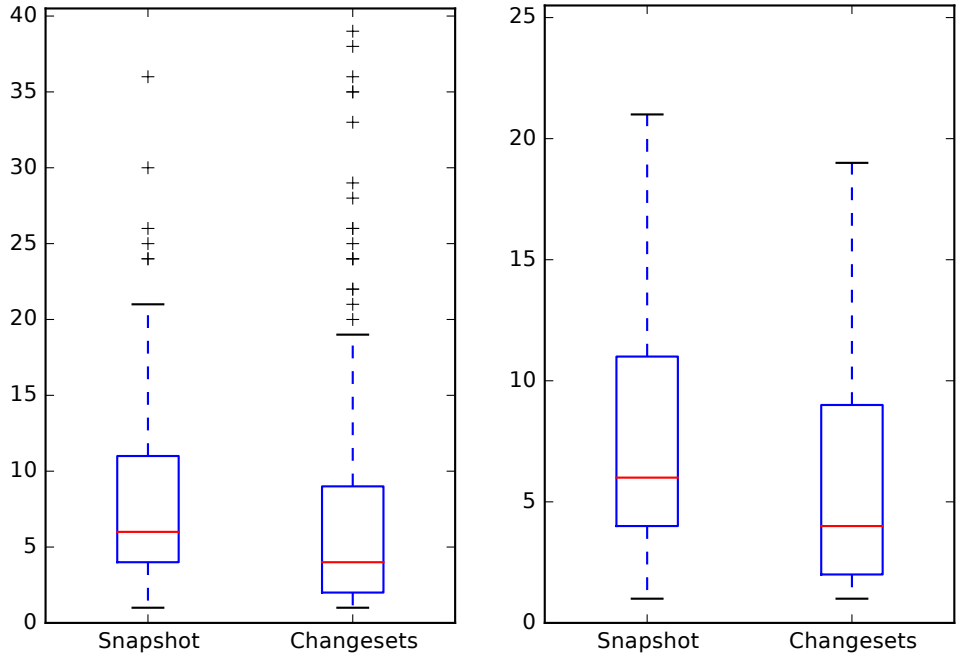
B. RP2: DEVELOPER IDENTIFICATION BOXPLOTS

In these figures, we display the effectiveness measures for each subject system for *Research Problem 2* (RP2).



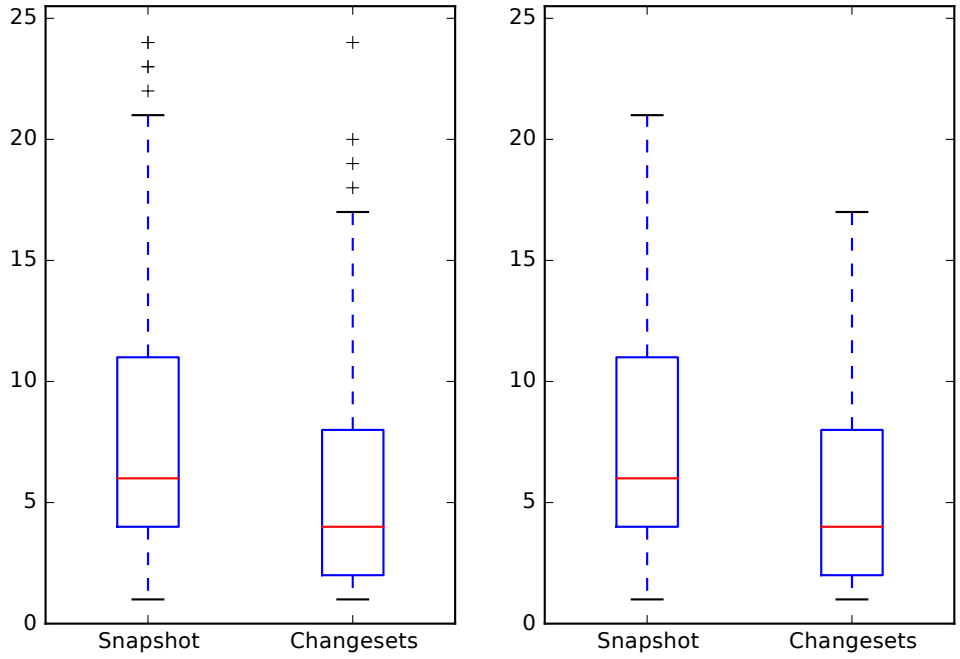
(a) Including outliers (b) Excluding outliers

Figure B.1: *RQ 2.1*: Developer Identification effectiveness measures for BookKeeper v4.3.0



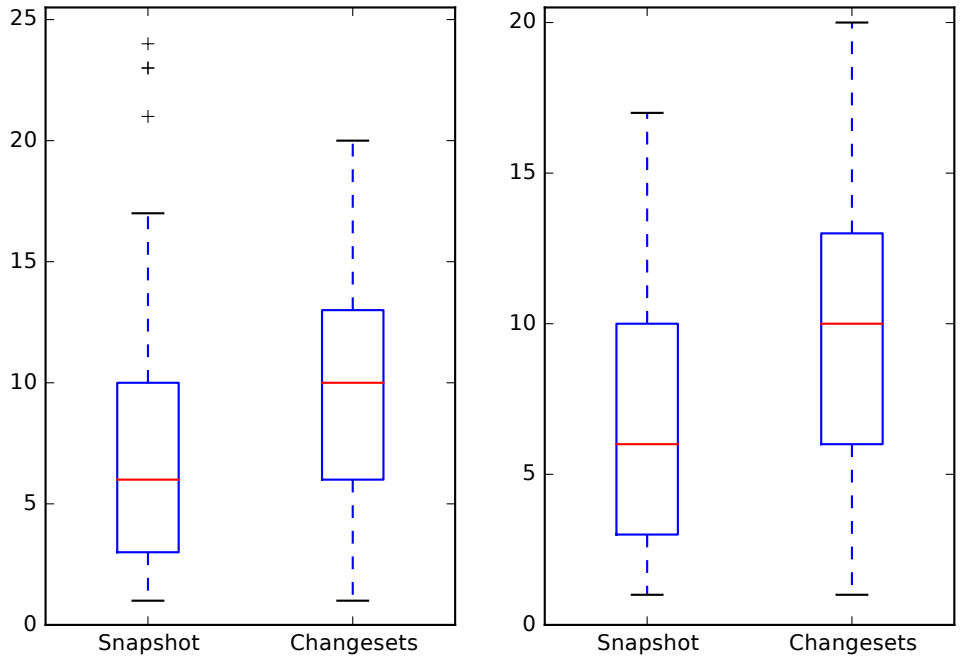
(a) Including outliers (b) Excluding outliers

Figure B.2: *RQ 2.1*: Developer Identification effectiveness measures for Mahout v0.10.0



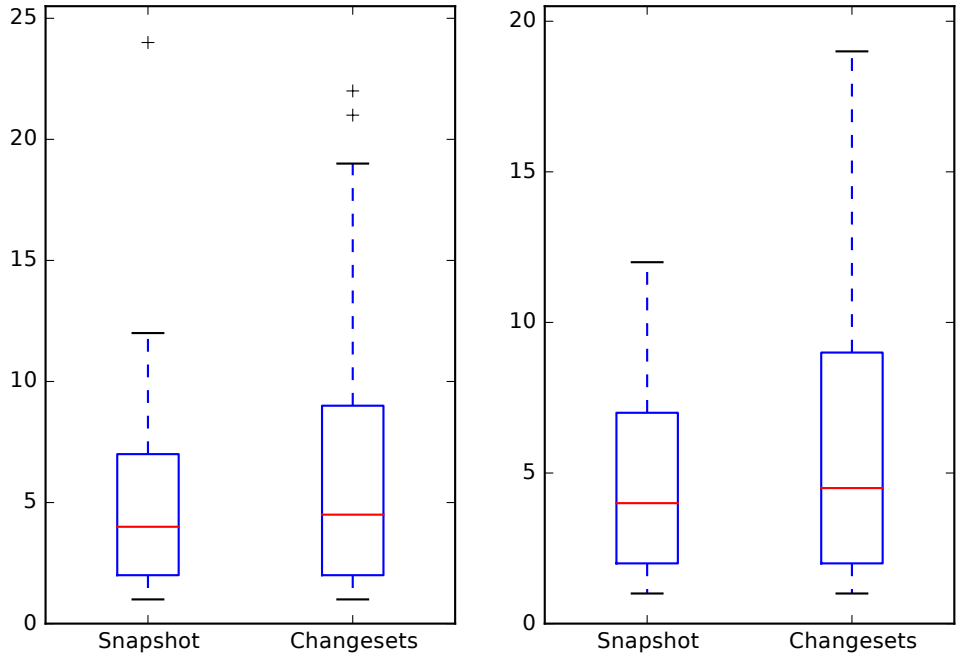
(a) Including outliers (b) Excluding outliers

Figure B.3: *RQ 2.1*: Developer Identification effectiveness measures for OpenJPA v2.3.0



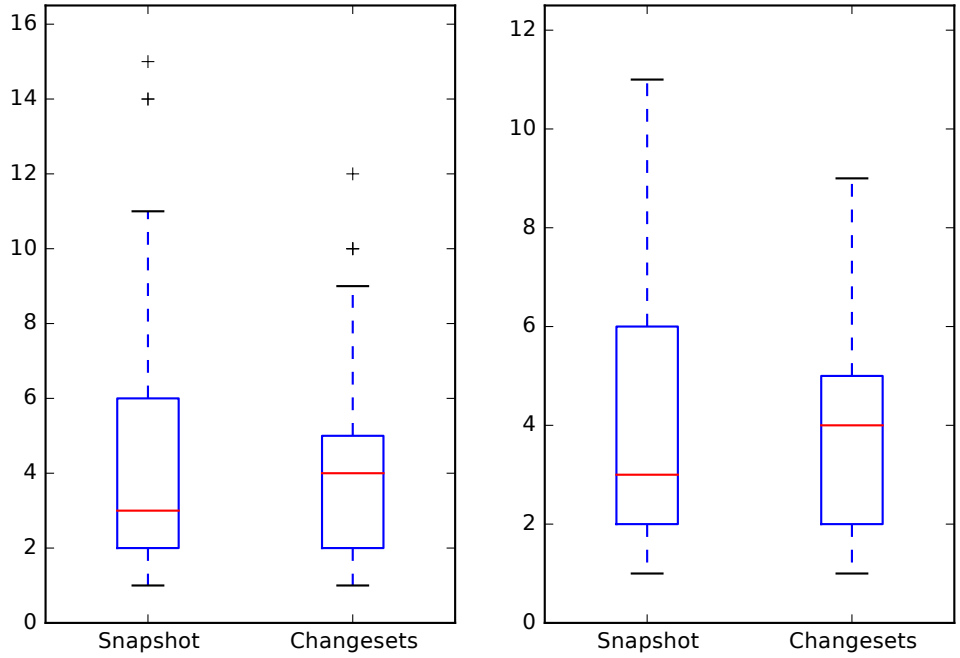
(a) Including outliers (b) Excluding outliers

Figure B.4: *RQ 2.1*: Developer Identification effectiveness measures for Pig v0.14.0



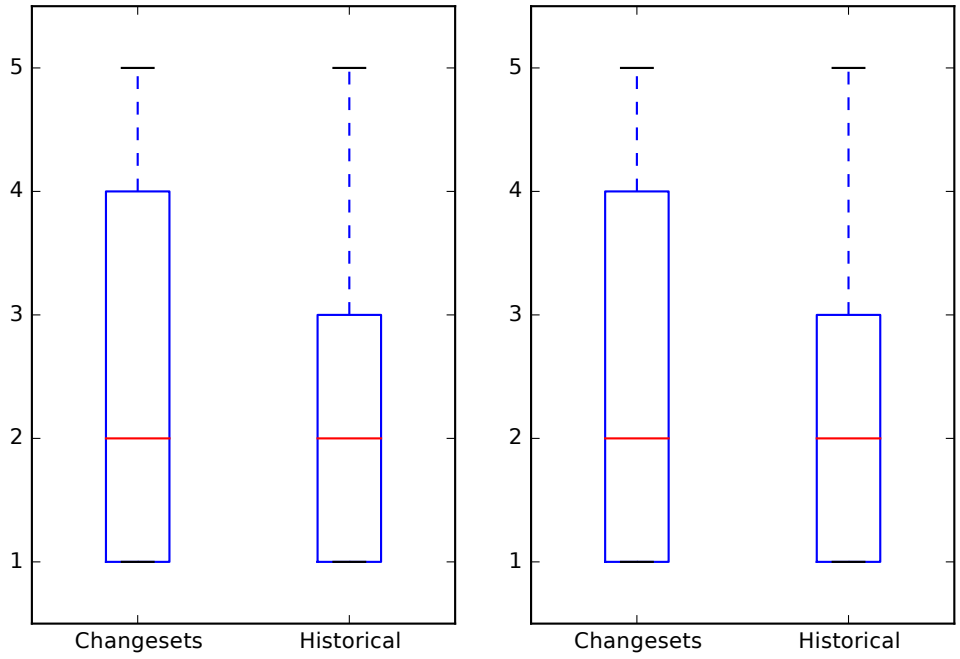
(a) Including outliers (b) Excluding outliers

Figure B.5: *RQ 2.1*: Developer Identification effectiveness measures for Tika v1.8



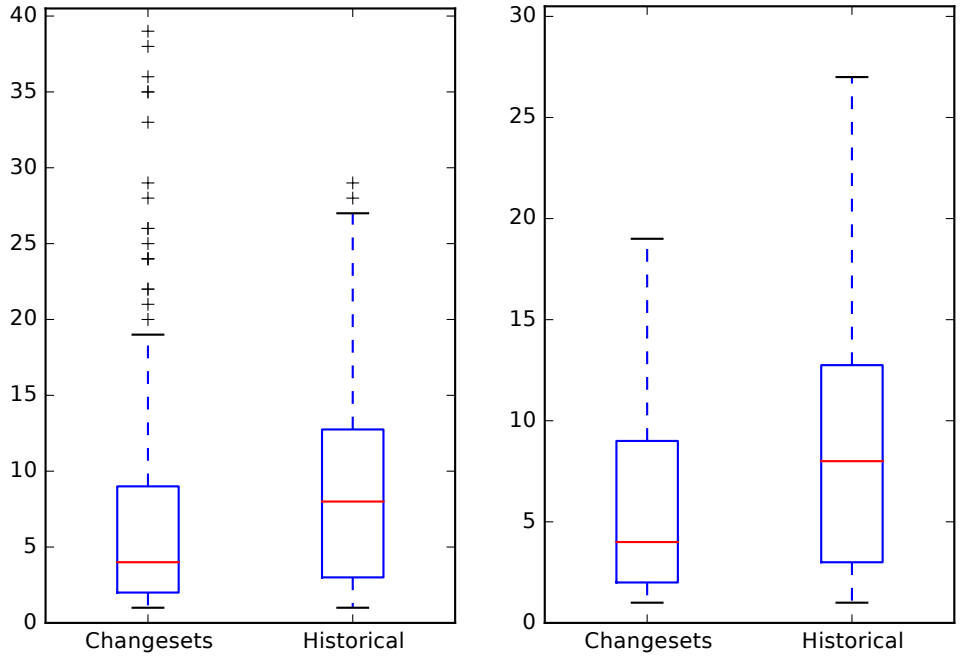
(a) Including outliers (b) Excluding outliers

Figure B.6: *RQ 2.1*: Developer Identification effectiveness measures for ZooKeeper v3.5.0



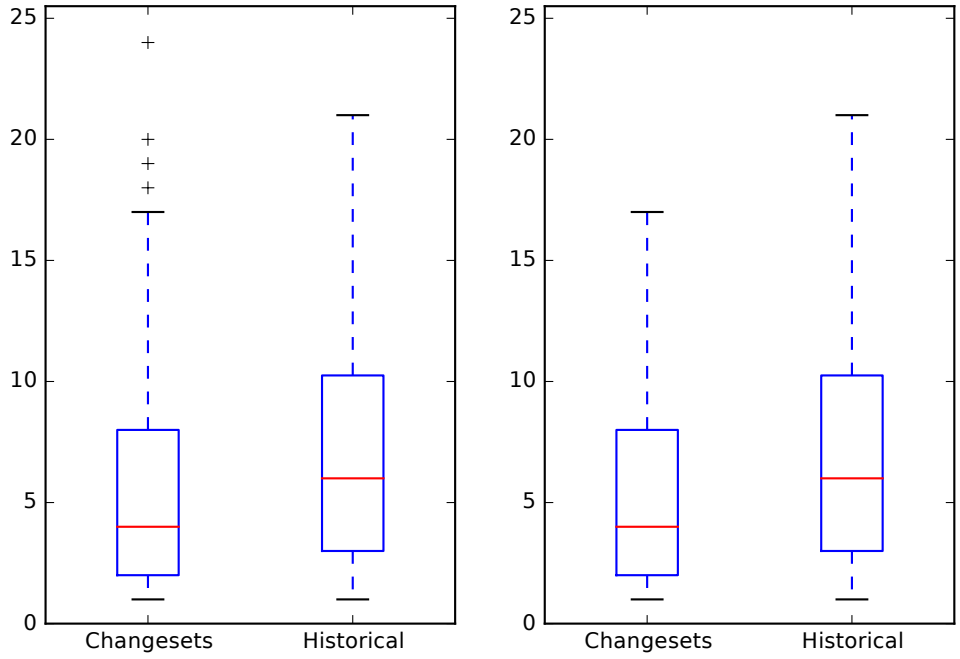
(a) Including outliers (b) Excluding outliers

Figure B.7: *RQ 2.2*: Developer Identification effectiveness measures for BookKeeper v4.3.0



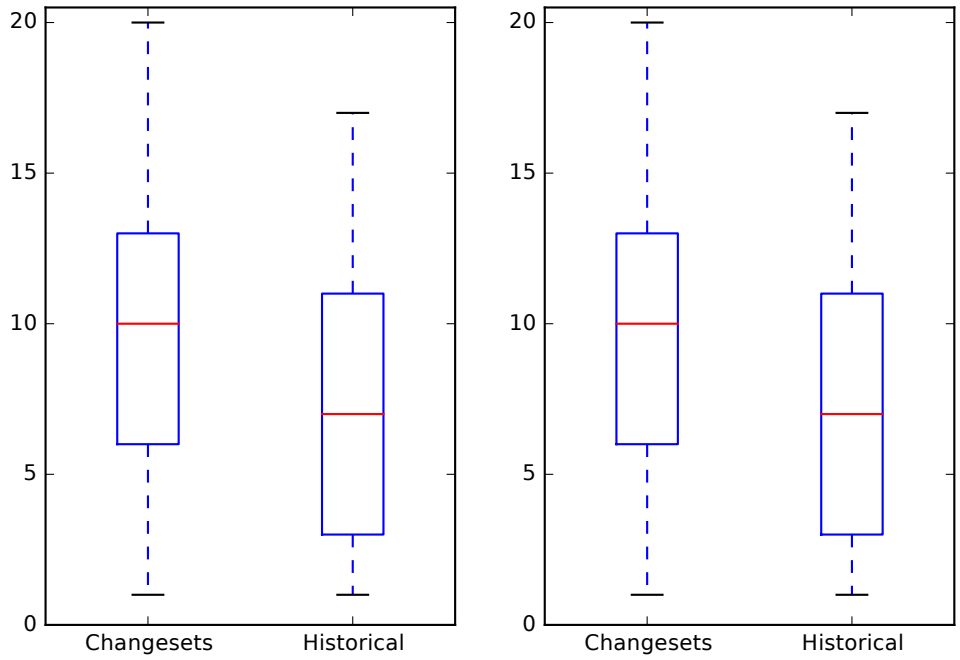
(a) Including outliers (b) Excluding outliers

Figure B.8: *RQ 2.2*: Developer Identification effectiveness measures for Mahout v0.10.0



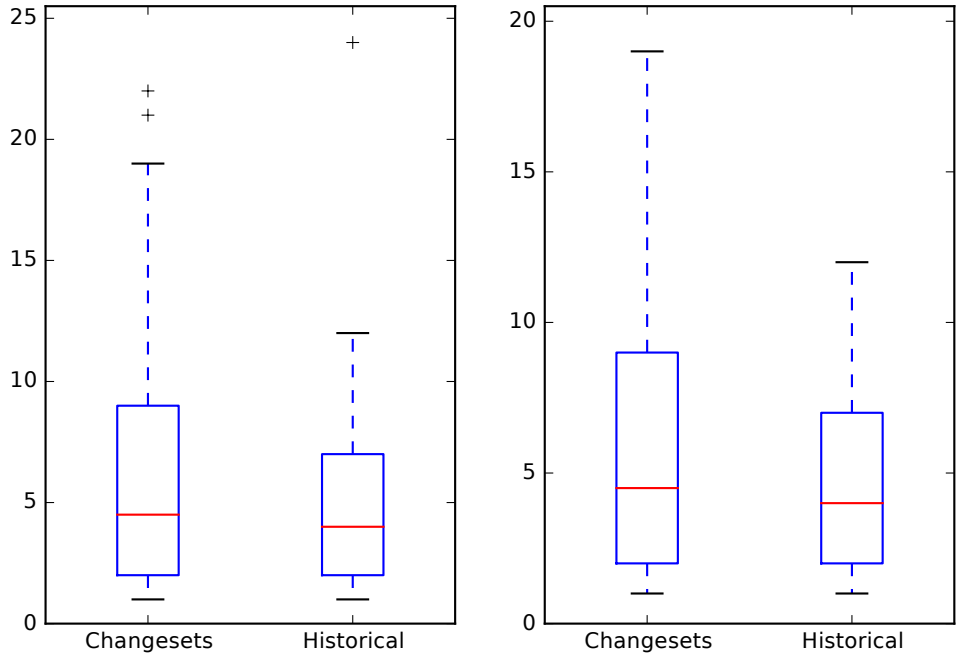
(a) Including outliers (b) Excluding outliers

Figure B.9: *RQ 2.2*: Developer Identification effectiveness measures for OpenJPA v2.3.0



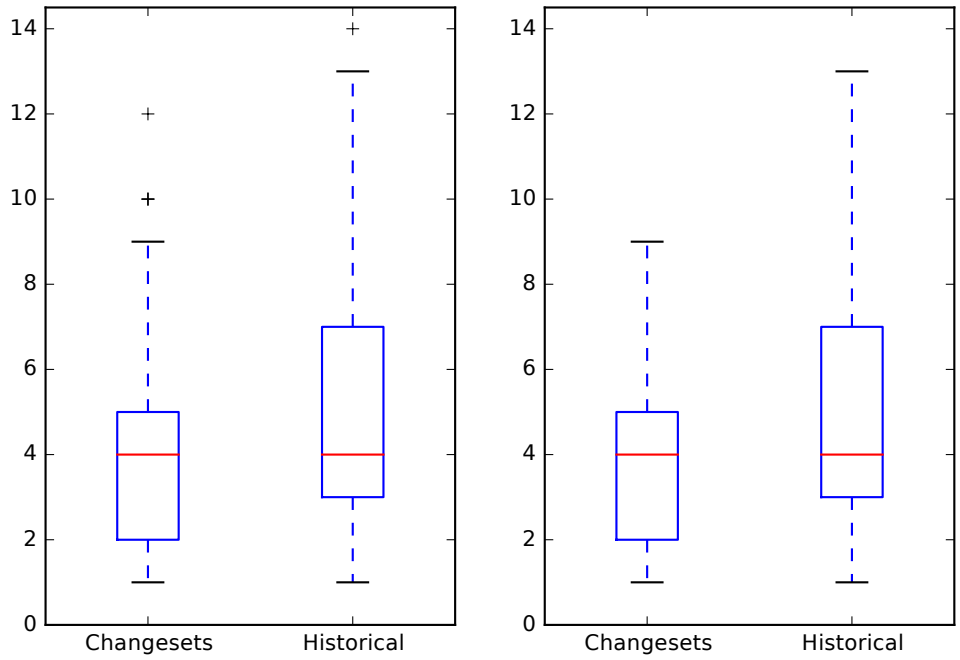
(a) Including outliers (b) Excluding outliers

Figure B.10: *RQ 2.2*: Developer Identification effectiveness measures for Pig v0.14.0



(a) Including outliers (b) Excluding outliers

Figure B.11: *RQ 2.2*: Developer Identification effectiveness measures for Tika v1.8



(a) Including outliers (b) Excluding outliers

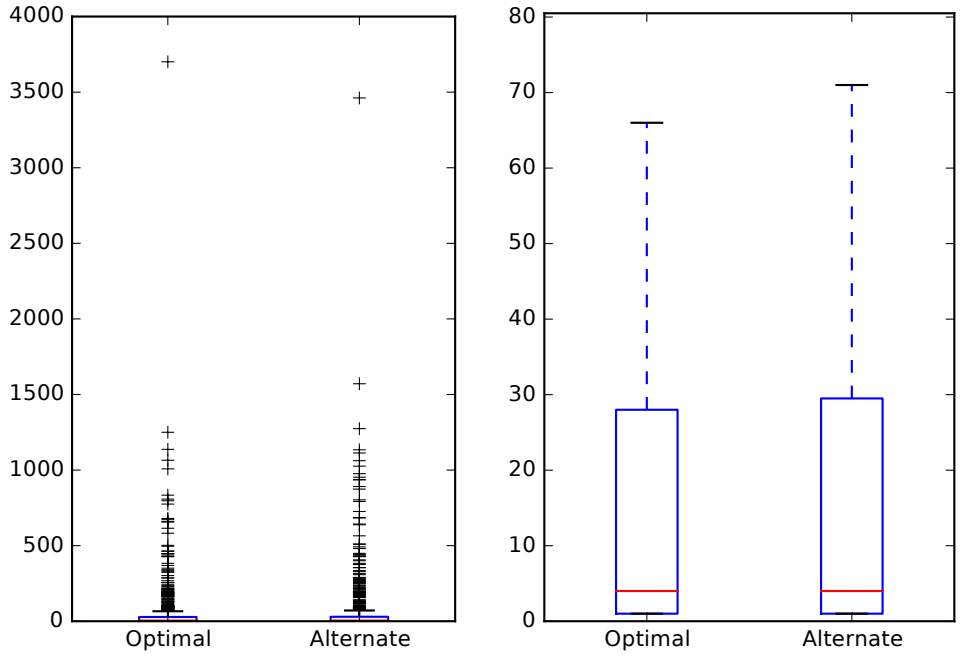
Figure B.12: *RQ 2.2*: Developer Identification effectiveness measures for ZooKeeper v3.5.0

C. RP3: MODEL CONFIGURATION SWEEP

In these tables, we bold entries where the MRR is highest for each of the two tasks and also highlight the row of the configuration used throughout this dissertation.

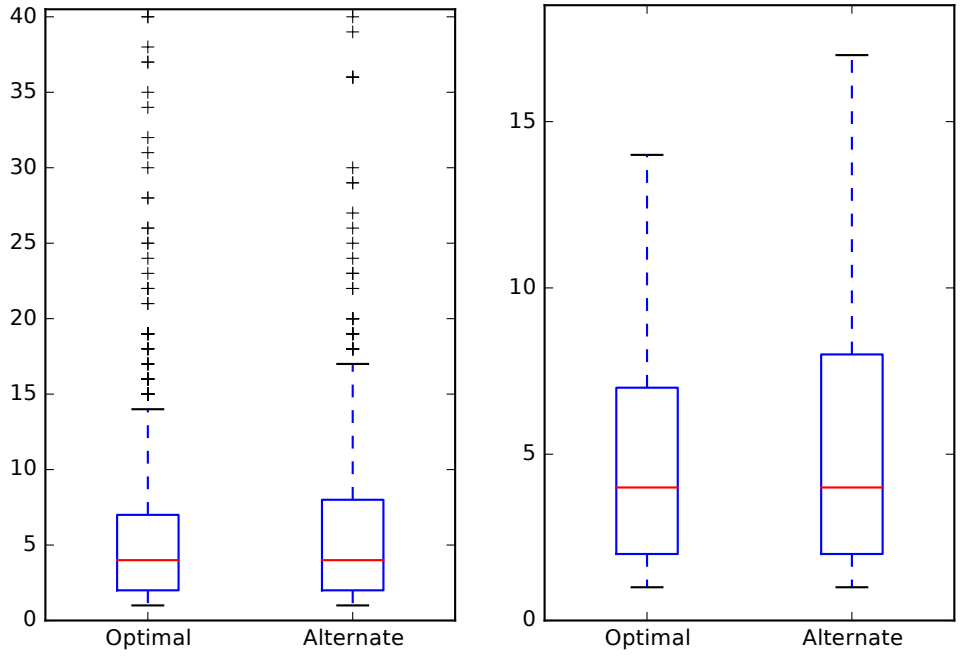
Table C.1: MRR values of all subject systems model construction sweep (*RQ 3.1*)

K	α	η	FLT	DIT
100	$1/K$	$1/K$	0.3165	0.3017
100	$1/K$	$2/K$	0.3116	0.2923
100	$1/K$	$5/K$	0.2984	0.2855
100	$1/K$	auto	0.3147	0.3031
100	$2/K$	$1/K$	0.3189	0.3007
100	$2/K$	$2/K$	0.3113	0.2919
100	$2/K$	$5/K$	0.3040	0.2849
100	$2/K$	auto	0.3151	0.3026
100	$5/K$	$1/K$	0.3137	0.3150
100	$5/K$	$2/K$	0.3159	0.2915
100	$5/K$	$5/K$	0.3042	0.2837
100	$5/K$	auto	0.3135	0.3139
100	auto	$1/K$	0.3155	0.3028
100	auto	$2/K$	0.3142	0.2923
100	auto	$5/K$	0.3010	0.2838
100	auto	auto	0.3151	0.3021
200	$1/K$	$1/K$	0.3671	0.3509
200	$1/K$	$2/K$	0.3827	0.3338
200	$1/K$	$5/K$	0.3746	0.3106
200	$1/K$	auto	0.3714	0.3434
200	$2/K$	$1/K$	0.3661	0.3462
200	$2/K$	$2/K$	0.3793	0.3282
200	$2/K$	$5/K$	0.3698	0.3143
200	$2/K$	auto	0.3715	0.3386
200	$5/K$	$1/K$	0.3635	0.3457
200	$5/K$	$2/K$	0.3713	0.3327
200	$5/K$	$5/K$	0.3687	0.3244
200	$5/K$	auto	0.3642	0.3412
200	auto	$1/K$	0.3669	0.3480
200	auto	$2/K$	0.3797	0.3359
200	auto	$5/K$	0.3736	0.3111
200	auto	auto	0.3706	0.3427
500	$1/K$	$1/K$	0.4092	0.3770
500	$1/K$	$2/K$	0.4090	0.3761
500	$1/K$	$5/K$	0.3837	0.3657
500	$1/K$	auto	0.4129	0.3742
500	$2/K$	$1/K$	0.4122	0.3718
500	$2/K$	$2/K$	0.4011	0.3802
500	$2/K$	$5/K$	0.3812	0.3691
500	$2/K$	auto	**0.4162**	0.3718
500	$5/K$	$1/K$	0.4064	0.3795
500	$5/K$	$2/K$	0.3996	0.3750
500	$5/K$	$5/K$	0.3850	0.3652
500	$5/K$	auto	0.4107	**0.3818**
500	auto	$1/K$	0.4101	0.3778
500	auto	$2/K$	0.4090	0.3759
500	auto	$5/K$	0.3842	0.3666
500	auto	auto	0.4126	0.3754



(a) Including outliers (b) Excluding outliers

Figure C.1: Feature Location effectiveness measures of optimal ($MRR = 0.4162$) and alternate ($MRR = 0.4107$) model configurations for all subject systems

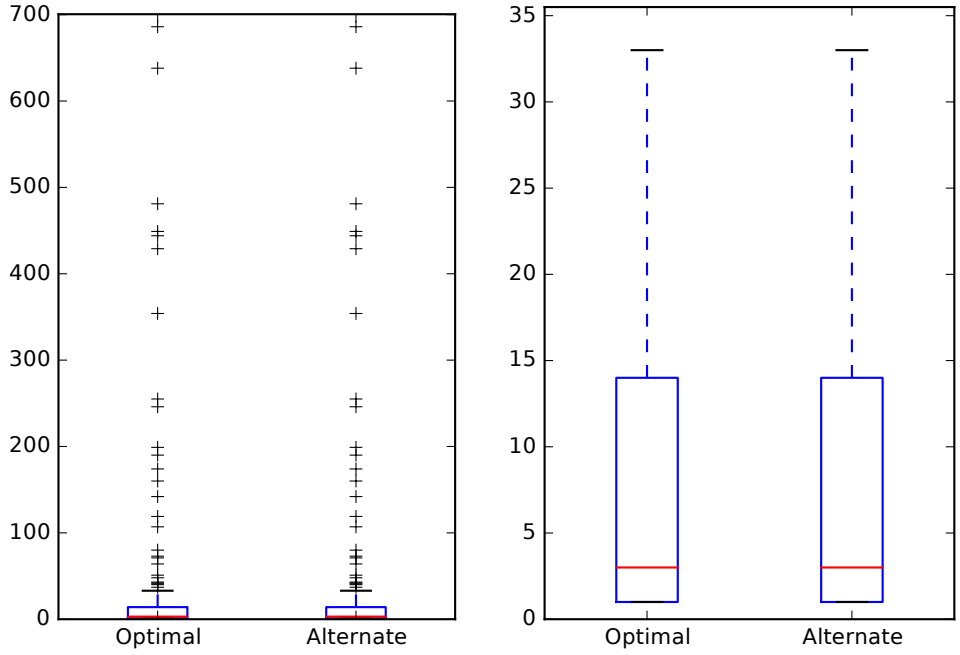


(a) Including outliers (b) Excluding outliers

Figure C.2: Developer Identification effectiveness measures of optimal ($MRR = 0.3818$) and alternate ($MRR = 0.3718$) model configurations for all subject systems

Table C.2: MRR values of BookKeeper v4.3.0 model construction sweep (*RQ 3.1*)

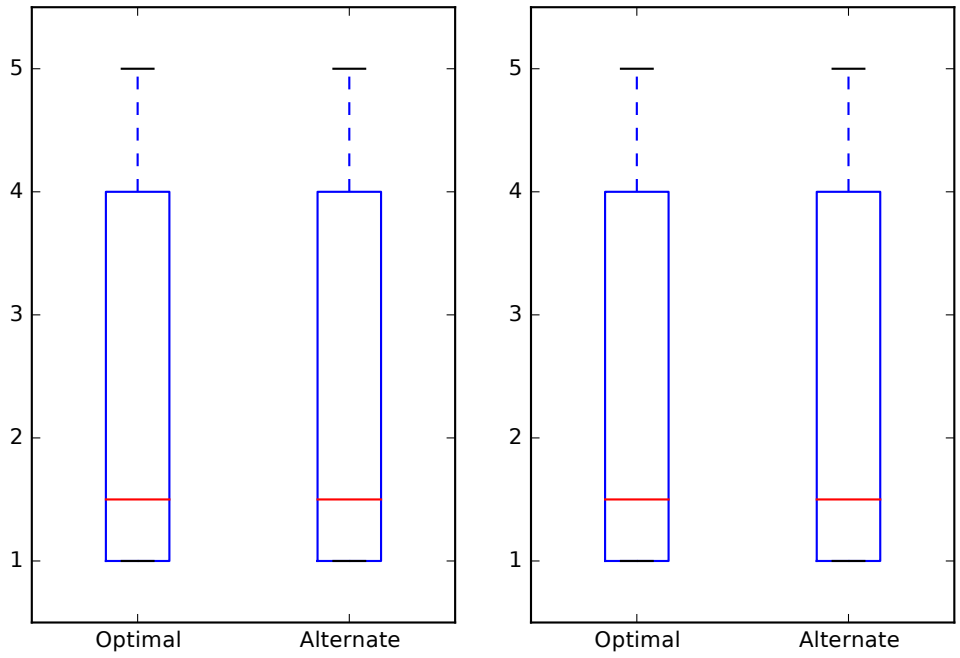
K	α	η	FLT	DIT
100	$1/K$	$1/K$	0.4143	0.5129
100	$1/K$	$2/K$	0.4002	0.4740
100	$1/K$	$5/K$	0.3420	0.4296
100	$1/K$	auto	0.3982	0.5051
100	$2/K$	$1/K$	0.4208	0.5097
100	$2/K$	$2/K$	0.3963	0.4755
100	$2/K$	$5/K$	0.3611	0.4392
100	$2/K$	auto	0.3967	0.5176
100	$5/K$	$1/K$	0.4194	0.5374
100	$5/K$	$2/K$	0.3808	0.4892
100	$5/K$	$5/K$	0.3720	0.4443
100	$5/K$	auto	0.4123	0.5321
100	auto	$1/K$	0.4108	0.5160
100	auto	$2/K$	0.4069	0.4780
100	auto	$5/K$	0.3420	0.4296
100	auto	auto	0.4009	0.5051
200	$1/K$	$1/K$	0.4422	0.5945
200	$1/K$	$2/K$	0.4468	0.5557
200	$1/K$	$5/K$	0.4711	0.4902
200	$1/K$	auto	0.4566	0.5566
200	$2/K$	$1/K$	0.4349	0.5686
200	$2/K$	$2/K$	0.4434	0.5595
200	$2/K$	$5/K$	0.4638	0.5114
200	$2/K$	auto	0.4501	0.5571
200	$5/K$	$1/K$	0.4237	0.5823
200	$5/K$	$2/K$	0.4279	0.5805
200	$5/K$	$5/K$	0.4770	0.5223
200	$5/K$	auto	0.4333	0.5554
200	auto	$1/K$	0.4387	0.5945
200	auto	$2/K$	0.4256	0.5652
200	auto	$5/K$	0.4720	0.4902
200	auto	auto	0.4566	0.5566
500	$1/K$	$1/K$	0.4567	0.6513
500	$1/K$	$2/K$	0.4733	0.6470
500	$1/K$	$5/K$	0.4572	0.6189
500	$1/K$	auto	0.4711	0.6269
500	$2/K$	$1/K$	0.4618	0.6419
500	$2/K$	$2/K$	0.4621	0.6419
500	$2/K$	$5/K$	0.4497	0.6226
500	$2/K$	auto	0.4633	0.6459
500	$5/K$	$1/K$	0.4740	0.6571
500	$5/K$	$2/K$	0.4826	0.6432
500	$5/K$	$5/K$	0.4400	0.6327
500	$5/K$	auto	**0.4884**	**0.6642**
500	auto	$1/K$	0.4578	0.6513
500	auto	$2/K$	0.4720	0.6470
500	auto	$5/K$	0.4567	0.6189
500	auto	auto	0.4710	0.6266



(a) Including outliers

(b) Excluding outliers

Figure C.3: Feature Location effectiveness measures of optimal ($MRR = 0.4884$) and alternate ($MRR = 0.4884$) model configurations for BookKeeper v4.3.0



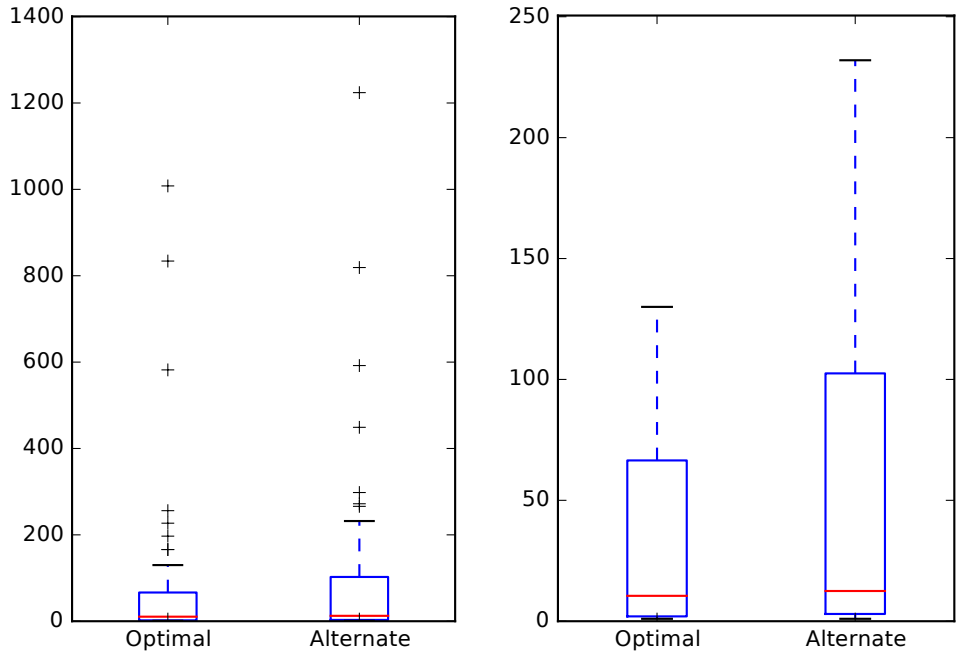
(a) Including outliers

(b) Excluding outliers

Figure C.4: Developer Identification effectiveness measures of optimal ($MRR = 0.6642$) and alternate ($MRR = 0.6642$) model configurations for BookKeeper v4.3.0

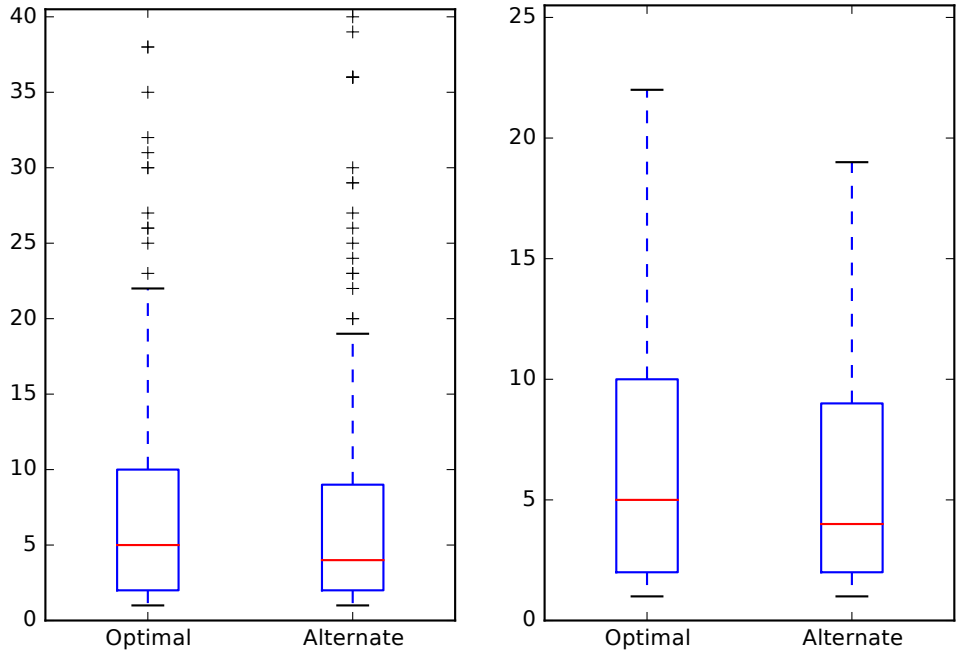
Table C.3: MRR values of Mahout v0.10.0 model construction sweep (*RQ 3.1*)

K	α	η	FLT	DIT
100	$1/K$	$1/K$	0.2536	0.2501
100	$1/K$	$2/K$	0.2601	0.2478
100	$1/K$	$5/K$	0.2830	0.2560
100	$1/K$	auto	0.2427	0.2594
100	$2/K$	$1/K$	0.2687	0.2395
100	$2/K$	$2/K$	0.2504	0.2518
100	$2/K$	$5/K$	0.2952	0.2556
100	$2/K$	auto	0.2579	0.2370
100	$5/K$	$1/K$	0.2244	0.2686
100	$5/K$	$2/K$	0.2641	0.2400
100	$5/K$	$5/K$	0.2940	0.2401
100	$5/K$	auto	0.2355	0.2686
100	auto	$1/K$	0.2436	0.2501
100	auto	$2/K$	0.2601	0.2478
100	auto	$5/K$	0.2830	0.2563
100	auto	auto	0.2404	0.2600
200	$1/K$	$1/K$	0.2846	0.2806
200	$1/K$	$2/K$	0.3193	0.2950
200	$1/K$	$5/K$	0.2751	0.2855
200	$1/K$	auto	0.2987	0.2788
200	$2/K$	$1/K$	0.2667	0.2811
200	$2/K$	$2/K$	0.3183	0.2787
200	$2/K$	$5/K$	0.2725	0.2872
200	$2/K$	auto	0.2767	0.2856
200	$5/K$	$1/K$	0.2542	0.2740
200	$5/K$	$2/K$	0.2904	0.2764
200	$5/K$	$5/K$	0.2837	0.2908
200	$5/K$	auto	0.2602	0.2749
200	auto	$1/K$	0.2879	0.2812
200	auto	$2/K$	0.3193	0.2954
200	auto	$5/K$	0.2760	0.2855
200	auto	auto	0.2987	0.2789
500	$1/K$	$1/K$	0.2730	0.3340
500	$1/K$	$2/K$	0.2860	0.3517
500	$1/K$	$5/K$	0.2802	**0.3544**
500	$1/K$	auto	0.2726	0.3404
500	$2/K$	$1/K$	0.3106	0.3337
500	$2/K$	$2/K$	0.2836	0.3488
500	$2/K$	$5/K$	0.2861	0.3251
500	$2/K$	auto	**0.3390**	0.3504
500	$5/K$	$1/K$	0.2579	0.3368
500	$5/K$	$2/K$	0.2437	0.3535
500	$5/K$	$5/K$	0.2502	0.3355
500	$5/K$	auto	0.2685	0.3381
500	auto	$1/K$	0.2733	0.3340
500	auto	$2/K$	0.2860	0.3517
500	auto	$5/K$	0.2802	0.3543
500	auto	auto	0.2727	0.3400



(a) Including outliers (b) Excluding outliers

Figure C.5: Feature Location effectiveness measures of optimal ($MRR = 0.3390$) and alternate ($MRR = 0.2802$) model configurations for Mahout v0.10.0

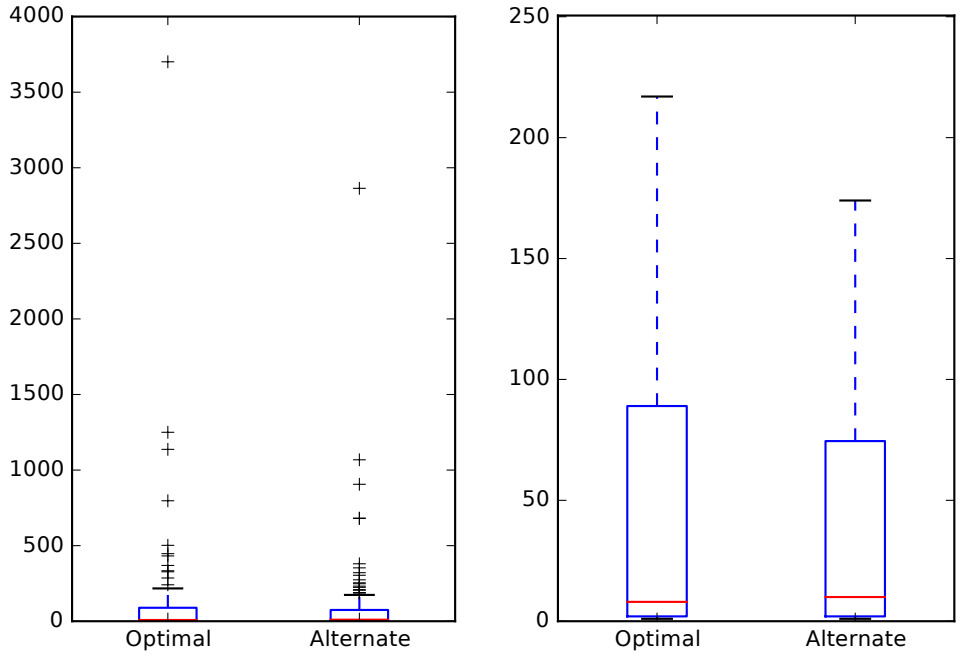


(a) Including outliers (b) Excluding outliers

Figure C.6: Developer Identification effectiveness measures of optimal ($MRR = 0.3544$) and alternate ($MRR = 0.3504$) model configurations for Mahout v0.10.0

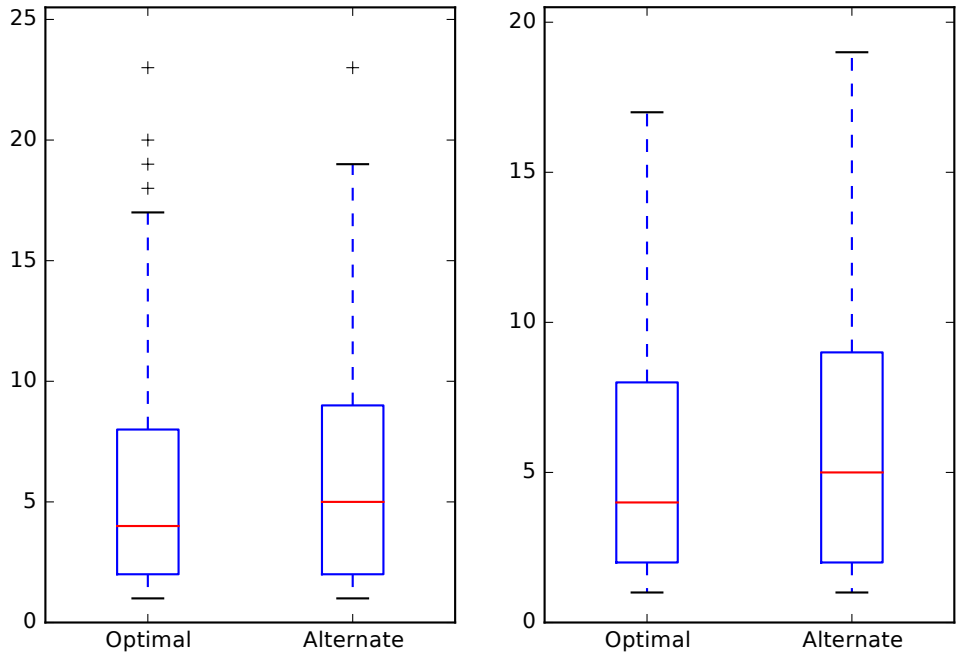
Table C.4: MRR values of OpenJPA v2.3.0 model construction sweep (*RQ 3.1*)

K	α	η	FLT	DIT
100	$1/K$	$1/K$	0.1998	0.2654
100	$1/K$	$2/K$	0.2045	0.2705
100	$1/K$	$5/K$	0.1953	0.2716
100	$1/K$	auto	0.2038	0.2697
100	$2/K$	$1/K$	0.2126	0.2664
100	$2/K$	$2/K$	0.2006	0.2697
100	$2/K$	$5/K$	0.2096	0.2675
100	$2/K$	auto	0.2097	0.2662
100	$5/K$	$1/K$	0.1819	0.2684
100	$5/K$	$2/K$	0.2052	0.2547
100	$5/K$	$5/K$	0.2020	0.2594
100	$5/K$	auto	0.1812	0.2665
100	auto	$1/K$	0.1998	0.2696
100	auto	$2/K$	0.2039	0.2670
100	auto	$5/K$	0.2051	0.2708
100	auto	auto	0.2025	0.2718
200	$1/K$	$1/K$	0.2733	0.3500
200	$1/K$	$2/K$	0.2688	0.3377
200	$1/K$	$5/K$	0.2706	0.3206
200	$1/K$	auto	0.2719	0.3489
200	$2/K$	$1/K$	0.2640	0.3342
200	$2/K$	$2/K$	0.2684	0.3260
200	$2/K$	$5/K$	0.2638	0.3410
200	$2/K$	auto	0.2619	0.3305
200	$5/K$	$1/K$	0.2771	0.3208
200	$5/K$	$2/K$	0.2578	0.3391
200	$5/K$	$5/K$	0.2558	0.3467
200	$5/K$	auto	0.2690	0.3320
200	auto	$1/K$	0.2758	0.3263
200	auto	$2/K$	0.2695	0.3383
200	auto	$5/K$	0.2691	0.3222
200	auto	auto	0.2719	0.3446
500	$1/K$	$1/K$	0.2989	0.3648
500	$1/K$	$2/K$	0.2837	0.3258
500	$1/K$	$5/K$	0.2655	0.3578
500	$1/K$	auto	0.3002	0.3647
500	$2/K$	$1/K$	0.2945	0.3394
500	$2/K$	$2/K$	0.2838	0.3373
500	$2/K$	$5/K$	0.2649	0.3618
500	$2/K$	auto	**0.3089**	0.3466
500	$5/K$	$1/K$	0.2895	0.3622
500	$5/K$	$2/K$	0.2543	0.3420
500	$5/K$	$5/K$	0.2885	0.3456
500	$5/K$	auto	0.2874	0.3644
500	auto	$1/K$	0.2983	**0.3695**
500	auto	$2/K$	0.2846	0.3253
500	auto	$5/K$	0.2721	0.3569
500	auto	auto	0.3002	0.3681



(a) Including outliers (b) Excluding outliers

Figure C.7: Feature Location effectiveness measures of optimal ($MRR = 0.3089$) and alternate ($MRR = 0.2983$) model configurations for OpenJPA v2.3.0

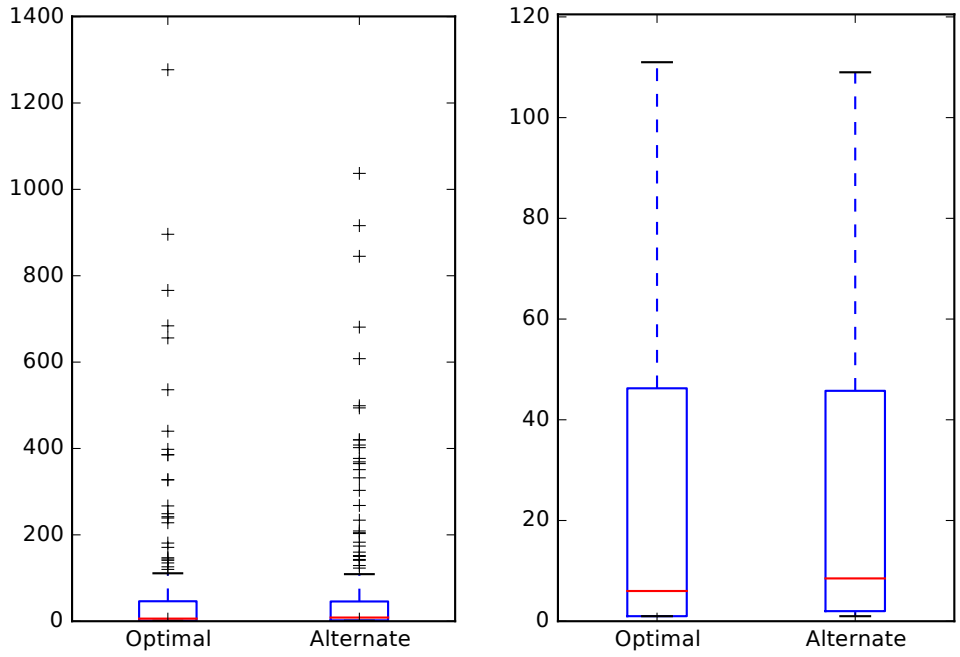


(a) Including outliers (b) Excluding outliers

Figure C.8: Developer Identification effectiveness measures of optimal ($MRR = 0.3695$) and alternate ($MRR = 0.3466$) model configurations for OpenJPA v2.3.0

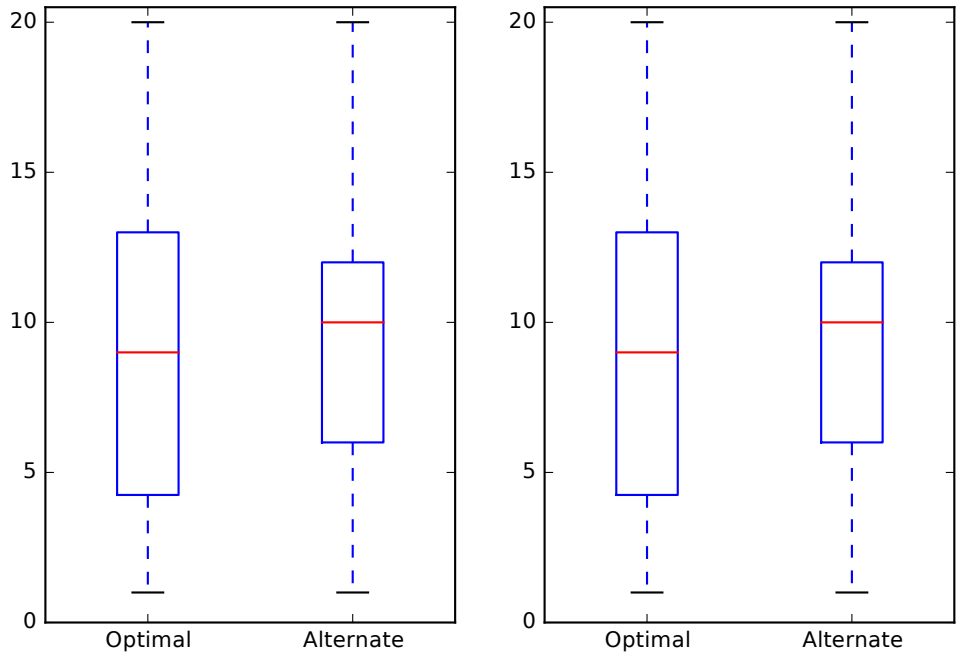
Table C.5: MRR values of Pig v0.14.0 model construction sweep (*RQ 3.1*)

K	α	η	FLT	DIT
100	$1/K$	$1/K$	0.2412	0.1960
100	$1/K$	$2/K$	0.2272	0.1859
100	$1/K$	$5/K$	0.2376	0.1931
100	$1/K$	auto	0.2411	0.1957
100	$2/K$	$1/K$	0.2338	0.1880
100	$2/K$	$2/K$	0.2283	0.1844
100	$2/K$	$5/K$	0.2370	0.1973
100	$2/K$	auto	0.2330	0.1889
100	$5/K$	$1/K$	0.2276	0.2018
100	$5/K$	$2/K$	0.2408	0.1813
100	$5/K$	$5/K$	0.2467	0.1902
100	$5/K$	auto	0.2277	0.1924
100	auto	$1/K$	0.2419	0.1961
100	auto	$2/K$	0.2263	0.1850
100	auto	$5/K$	0.2378	0.1925
100	auto	auto	0.2418	0.1968
200	$1/K$	$1/K$	0.2858	0.2060
200	$1/K$	$2/K$	0.3253	0.1811
200	$1/K$	$5/K$	0.3155	0.1730
200	$1/K$	auto	0.2862	0.2060
200	$2/K$	$1/K$	0.2931	0.2136
200	$2/K$	$2/K$	0.3132	0.1852
200	$2/K$	$5/K$	0.3120	0.1683
200	$2/K$	auto	0.3025	0.2081
200	$5/K$	$1/K$	0.2859	**0.2173**
200	$5/K$	$2/K$	0.3017	0.1878
200	$5/K$	$5/K$	0.3031	0.1882
200	$5/K$	auto	0.2906	0.2133
200	auto	$1/K$	0.2863	0.2058
200	auto	$2/K$	0.3249	0.1812
200	auto	$5/K$	0.3126	0.1724
200	auto	auto	0.2863	0.2079
500	$1/K$	$1/K$	0.3930	0.1759
500	$1/K$	$2/K$	**0.3964**	0.1631
500	$1/K$	$5/K$	0.3256	0.1775
500	$1/K$	auto	0.3898	0.1722
500	$2/K$	$1/K$	0.3905	0.1747
500	$2/K$	$2/K$	0.3821	0.1680
500	$2/K$	$5/K$	0.3244	0.1900
500	$2/K$	auto	0.3834	0.1760
500	$5/K$	$1/K$	0.3860	0.1710
500	$5/K$	$2/K$	0.3964	0.1609
500	$5/K$	$5/K$	0.3246	0.1701
500	$5/K$	auto	0.3859	0.1801
500	auto	$1/K$	0.3943	0.1762
500	auto	$2/K$	0.3936	0.1630
500	auto	$5/K$	0.3266	0.1746
500	auto	auto	0.3900	0.1744



(a) Including outliers (b) Excluding outliers

Figure C.9: Feature Location effectiveness measures of optimal ($MRR = 0.3964$) and alternate ($MRR = 0.2859$) model configurations for Pig v0.14.0

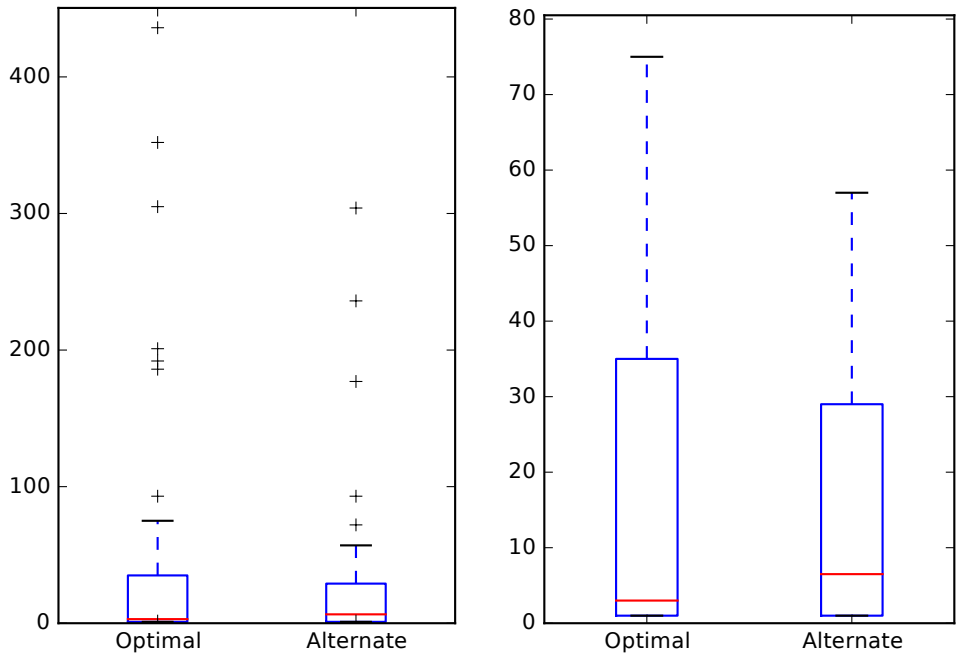


(a) Including outliers (b) Excluding outliers

Figure C.10: Developer Identification effectiveness measures of optimal ($MRR = 0.2173$) and alternate ($MRR = 0.1631$) model configurations for Pig v0.14.0

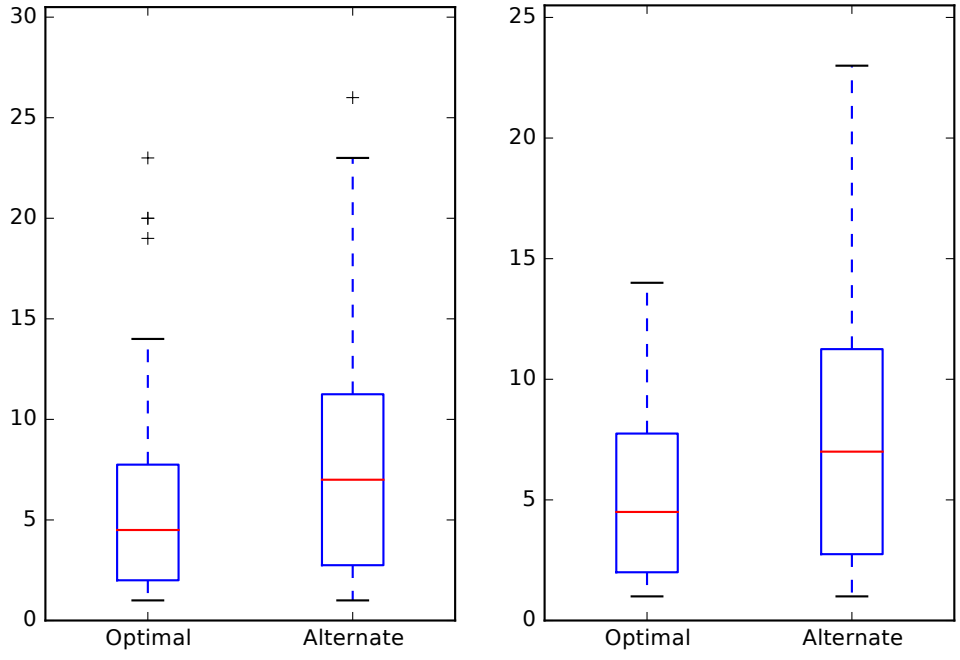
Table C.6: MRR values of Tika v1.8 model construction sweep (*RQ 3.1*)

K	α	η	FLT	DIT
100	$1/K$	$1/K$	0.3030	0.2813
100	$1/K$	$2/K$	0.3104	0.2858
100	$1/K$	$5/K$	0.3377	0.2377
100	$1/K$	auto	0.2793	0.2814
100	$2/K$	$1/K$	0.3013	0.2732
100	$2/K$	$2/K$	0.3007	0.2609
100	$2/K$	$5/K$	0.3418	0.2410
100	$2/K$	auto	0.2782	0.2881
100	$5/K$	$1/K$	0.3199	0.2898
100	$5/K$	$2/K$	0.3396	0.2799
100	$5/K$	$5/K$	0.3273	0.2223
100	$5/K$	auto	0.3244	0.2799
100	auto	$1/K$	0.3040	0.2926
100	auto	$2/K$	0.3126	0.2855
100	auto	$5/K$	0.3374	0.2372
100	auto	auto	0.2818	0.2809
200	$1/K$	$1/K$	0.4567	0.3135
200	$1/K$	$2/K$	0.4742	0.3164
200	$1/K$	$5/K$	0.3708	0.2828
200	$1/K$	auto	0.4747	0.3381
200	$2/K$	$1/K$	0.4583	0.3134
200	$2/K$	$2/K$	**0.4831**	0.3328
200	$2/K$	$5/K$	0.3906	0.2623
200	$2/K$	auto	0.4605	0.3140
200	$5/K$	$1/K$	0.4430	0.3677
200	$5/K$	$2/K$	0.4586	0.3103
200	$5/K$	$5/K$	0.4023	0.3060
200	$5/K$	auto	0.4171	0.3385
200	auto	$1/K$	0.4712	0.3216
200	auto	$2/K$	0.4743	0.3252
200	auto	$5/K$	0.3565	0.2824
200	auto	auto	0.4746	0.3382
500	$1/K$	$1/K$	0.4033	0.3609
500	$1/K$	$2/K$	0.3934	0.3572
500	$1/K$	$5/K$	0.4112	0.3320
500	$1/K$	auto	0.3944	0.3617
500	$2/K$	$1/K$	0.4364	0.3722
500	$2/K$	$2/K$	0.4004	0.3672
500	$2/K$	$5/K$	0.3922	**0.3775**
500	$2/K$	auto	0.4403	0.3597
500	$5/K$	$1/K$	0.3843	0.3570
500	$5/K$	$2/K$	0.3647	0.3720
500	$5/K$	$5/K$	0.3961	0.3565
500	$5/K$	auto	0.3868	0.3549
500	auto	$1/K$	0.4033	0.3608
500	auto	$2/K$	0.3934	0.3572
500	auto	$5/K$	0.3925	0.3747
500	auto	auto	0.3944	0.3634



(a) Including outliers (b) Excluding outliers

Figure C.11: Feature Location effectiveness measures of optimal ($MRR = 0.4831$) and alternate ($MRR = 0.3922$) model configurations for Tika v1.8

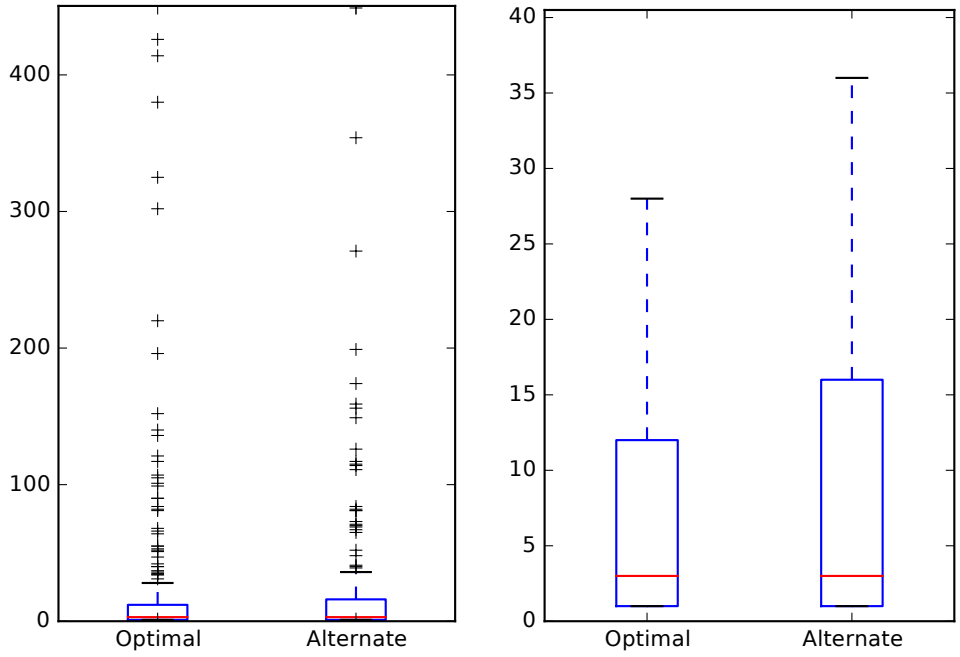


(a) Including outliers (b) Excluding outliers

Figure C.12: Developer Identification effectiveness measures of optimal ($MRR = 0.3775$) and alternate ($MRR = 0.3328$) model configurations for Tika v1.8

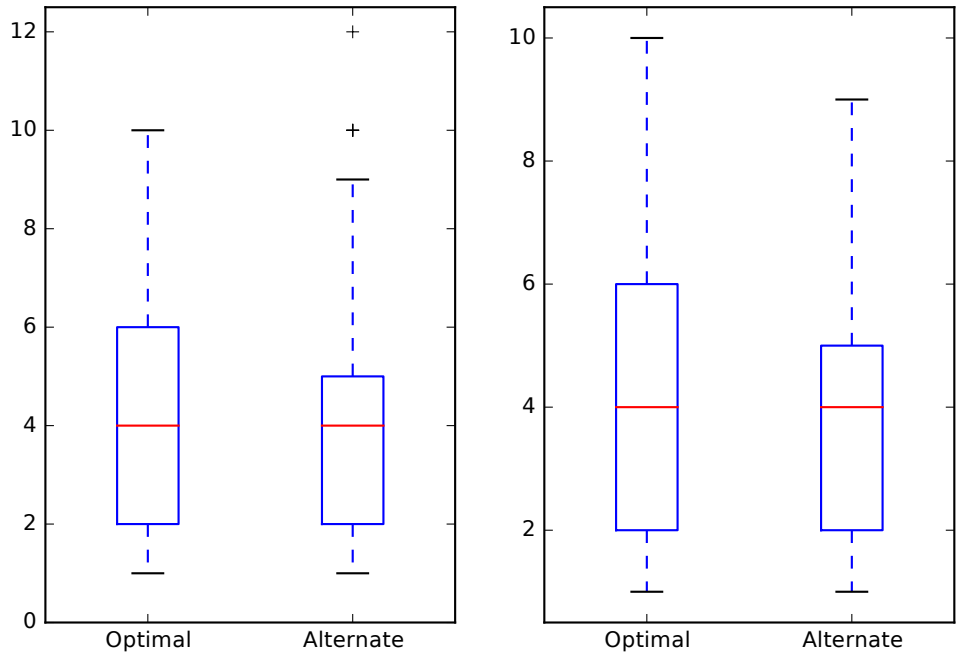
Table C.7: MRR values of ZooKeeper v3.5.0 model construction sweep (*RQ 3.1*)

K	α	η	FLT	DIT
100	$1/K$	$1/K$	0.3912	0.3058
100	$1/K$	$2/K$	0.3891	0.3005
100	$1/K$	$5/K$	0.3699	0.2984
100	$1/K$	auto	0.3986	0.3085
100	$2/K$	$1/K$	0.3906	0.3137
100	$2/K$	$2/K$	0.3953	0.3014
100	$2/K$	$5/K$	0.3660	0.2910
100	$2/K$	auto	0.4005	0.3144
100	$5/K$	$1/K$	0.4024	0.3212
100	$5/K$	$2/K$	0.3991	0.3037
100	$5/K$	$5/K$	0.3598	0.3005
100	$5/K$	auto	0.4033	0.3281
100	auto	$1/K$	0.3918	0.3048
100	auto	$2/K$	0.3940	0.3008
100	auto	$5/K$	0.3727	0.2940
100	auto	auto	0.3989	0.3040
200	$1/K$	$1/K$	0.4360	0.3598
200	$1/K$	$2/K$	0.4474	0.3417
200	$1/K$	$5/K$	0.4376	0.3223
200	$1/K$	auto	0.4363	0.3534
200	$2/K$	$1/K$	0.4405	0.3589
200	$2/K$	$2/K$	0.4463	0.3297
200	$2/K$	$5/K$	0.4306	0.3202
200	$2/K$	auto	0.4406	0.3451
200	$5/K$	$1/K$	0.4415	0.3506
200	$5/K$	$2/K$	0.4534	0.3302
200	$5/K$	$5/K$	0.4259	0.3241
200	$5/K$	auto	0.4418	0.3507
200	auto	$1/K$	0.4330	0.3592
200	auto	$2/K$	0.4505	0.3421
200	auto	$5/K$	0.4390	0.3234
200	auto	auto	0.4333	0.3517
500	$1/K$	$1/K$	0.4818	0.3985
500	$1/K$	$2/K$	0.4758	0.4145
500	$1/K$	$5/K$	0.4637	0.3775
500	$1/K$	auto	**0.4882**	0.4011
500	$2/K$	$1/K$	0.4799	0.3967
500	$2/K$	$2/K$	0.4670	**0.4213**
500	$2/K$	$5/K$	0.4630	0.3821
500	$2/K$	auto	0.4826	0.3865
500	$5/K$	$1/K$	0.4789	0.4066
500	$5/K$	$2/K$	0.4693	0.4057
500	$5/K$	$5/K$	0.4749	0.3831
500	$5/K$	auto	0.4824	0.4033
500	auto	$1/K$	0.4835	0.3987
500	auto	$2/K$	0.4783	0.4140
500	auto	$5/K$	0.4641	0.3773
500	auto	auto	0.4870	0.4021



(a) Including outliers (b) Excluding outliers

Figure C.13: Feature Location effectiveness measures of optimal ($MRR = 0.4882$) and alternate ($MRR = 0.4670$) model configurations for ZooKeeper v3.5.0



(a) Including outliers (b) Excluding outliers

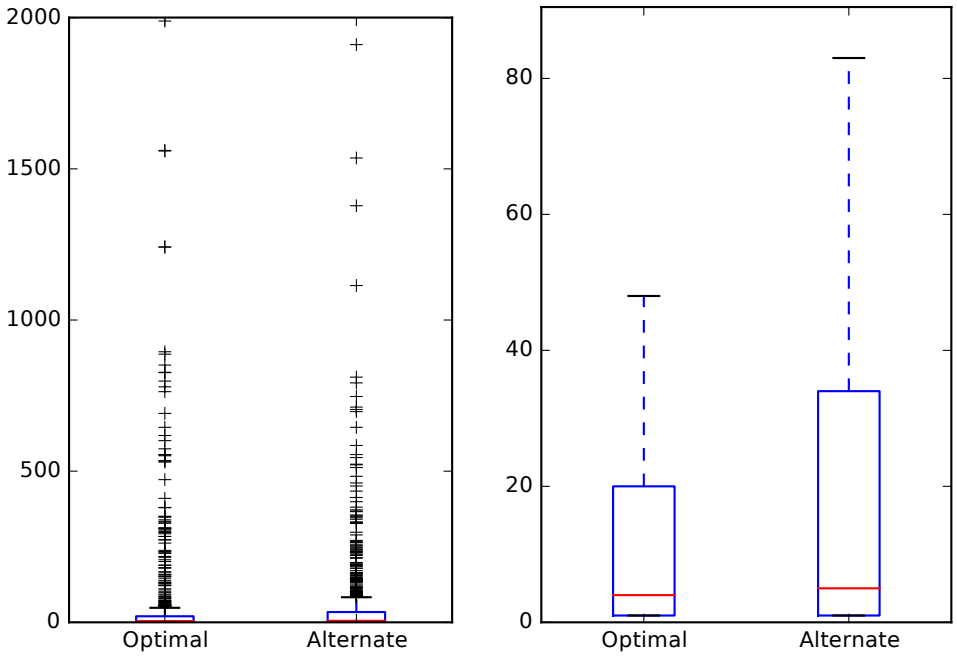
Figure C.14: Developer Identification effectiveness measures of optimal ($MRR = 0.4213$) and alternate ($MRR = 0.4011$) model configurations for ZooKeeper v3.5.0

D. RP3: CORPUS CONSTRUCTION SWEEP

In these tables, we bold entries where the MRR is highest for each of the two tasks and also highlight the row of the configuration used throughout this dissertation.

Table D.1: MRR values of all subject systems corpus construction sweep (*RQ 3.1* and *RQ 3.2*)

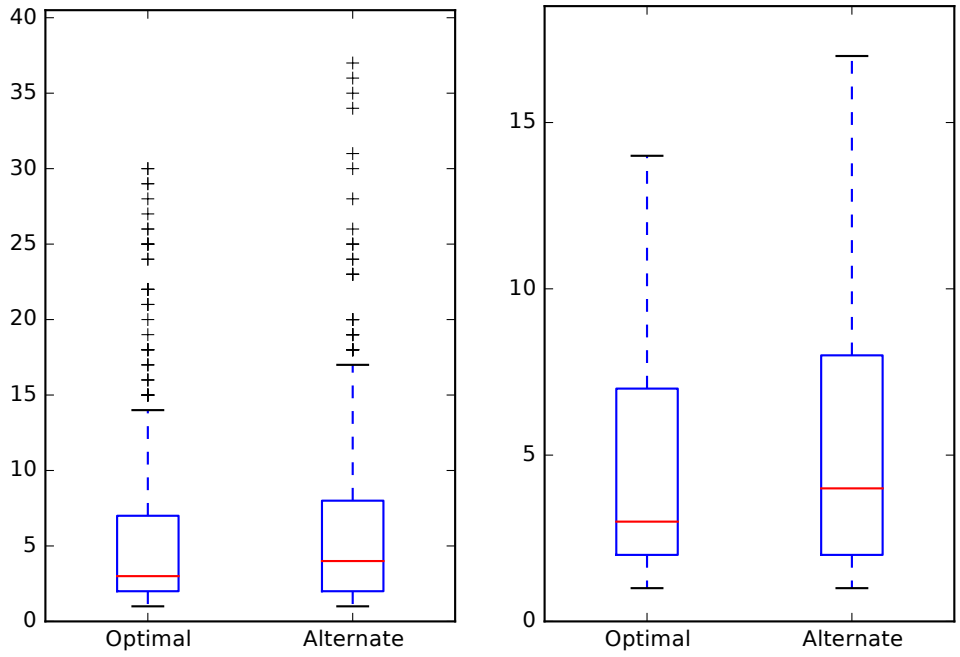
Configuration	FLT	DIT
(A,R,C,M)	0.4315	0.3784
(A,R,C)	0.4092	0.3770
(A,R,M)	0.4119	0.3646
(A,R)	0.4214	0.3462
(A,C,M)	**0.4517**	0.3785
(A,C)	0.4010	0.3802
(A,M)	0.4147	0.3537
(A)	0.4031	0.3380
(R,C,M)	0.4013	0.3391
(R,C)	0.3443	0.3373
(R,M)	0.3488	0.3308
(R)	0.3151	0.3147
(C,M)	0.3971	**0.4165**
(C)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



(a) Including outliers

(b) Excluding outliers

Figure D.1: Feature Location effectiveness measures of optimal ($MRR = 0.4517$) and alternate ($MRR = 0.3971$) corpus configurations for all subject systems



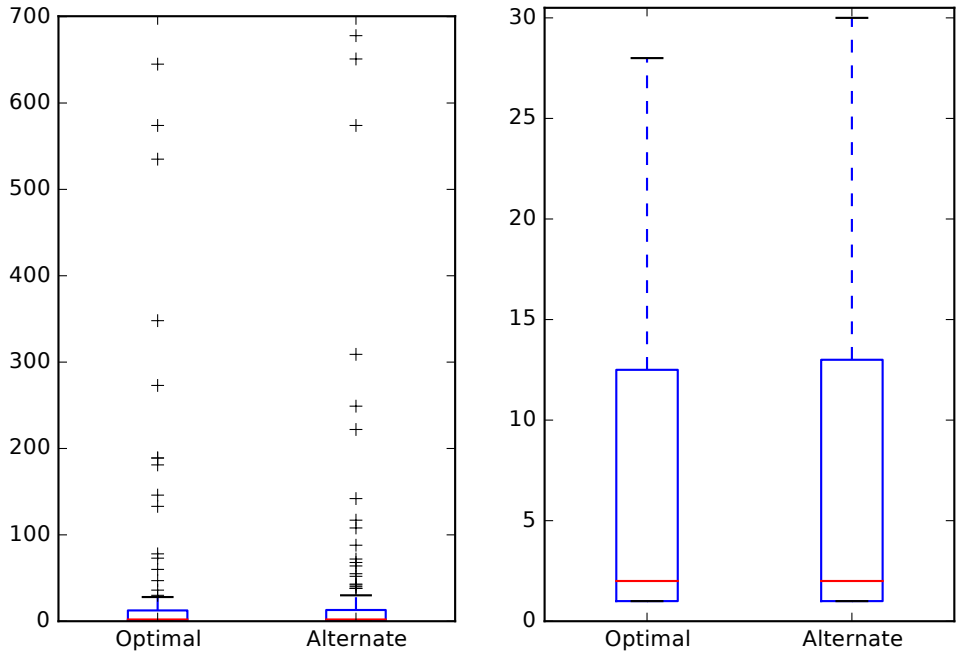
(a) Including outliers

(b) Excluding outliers

Figure D.2: Developer Identification effectiveness measures of optimal ($MRR = 0.4165$) and alternate ($MRR = 0.3785$) corpus configurations for all subject systems

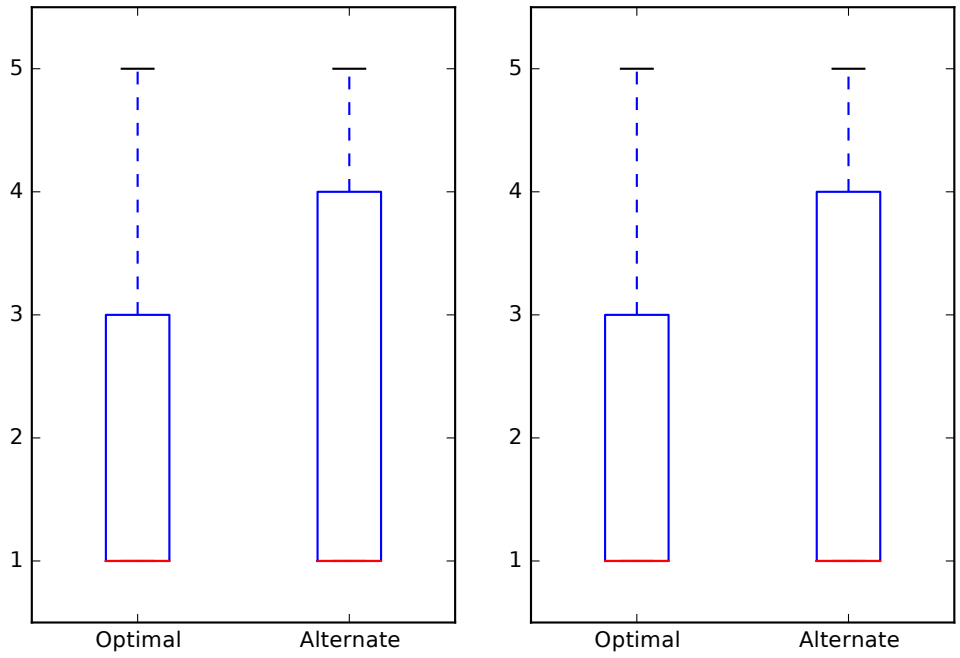
Table D.2: MRR values of BookKeeper v4.3.0 corpus construction sweep (*RQ 3.1* and *RQ 3.2*)

Configuration	FLT	DIT
(A,R,C,M)	0.5246	**0.7216**
(A,R,C)	0.4567	0.6513
(A,R,M)	0.5316	0.6465
(A,R)	0.4970	0.5425
(A,C,M)	**0.5327**	0.6974
(A,C)	0.4450	0.6517
(A,M)	0.5260	0.6722
(A)	0.4881	0.4821
(R,C,M)	0.4786	0.6530
(R,C)	0.4036	0.6526
(R,M)	0.3978	0.6507
(R)	0.3195	0.6375
(C,M)	0.4968	0.7114
(C)	0.3366	0.7057
(M)	0.3826	0.5132



(a) Including outliers (b) Excluding outliers

Figure D.3: Feature Location effectiveness measures of optimal ($MRR = 0.5327$) and alternate ($MRR = 0.5246$) corpus configurations for BookKeeper v4.3.0

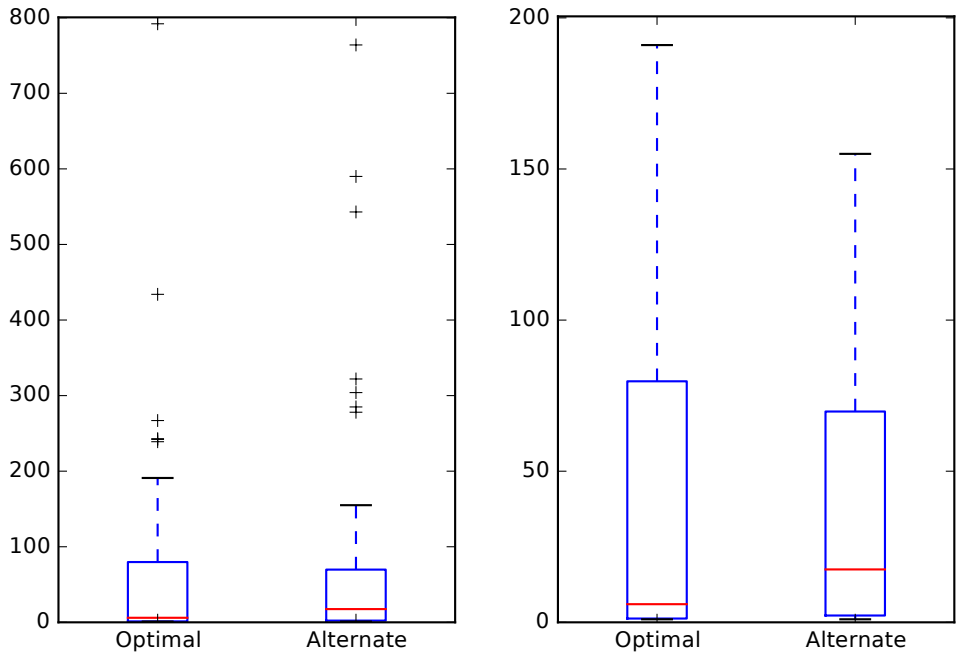


(a) Including outliers (b) Excluding outliers

Figure D.4: Developer Identification effectiveness measures of optimal ($MRR = 0.7216$) and alternate ($MRR = 0.6974$) corpus configurations for BookKeeper v4.3.0

Table D.3: MRR values of Mahout v0.10.0 corpus construction sweep (*RQ 3.1* and *RQ 3.2*)

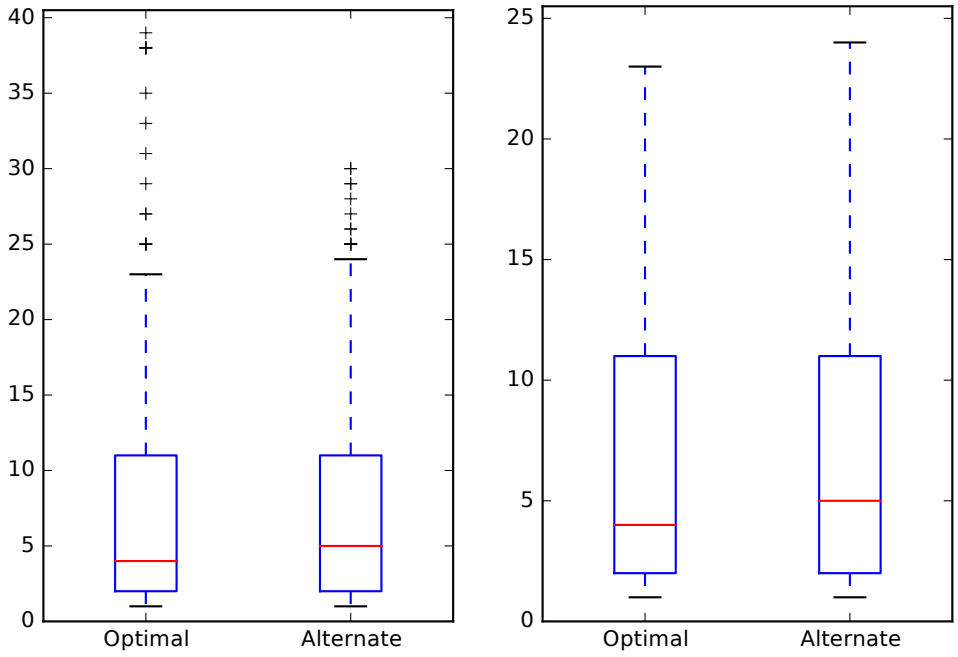
Configuration	FLT	DIT
(A,R,C,M)	0.3119	0.2785
(A,R,C)	0.2730	0.3340
(A,R,M)	0.2766	0.2797
(A,R)	0.2606	0.3052
(A,C,M)	0.3422	0.3124
(A,C)	0.3063	0.3492
(A,M)	0.2799	0.3116
(A)	0.2601	**0.3827**
(R,C,M)	0.3305	0.2505
(R,C)	0.3325	0.2719
(R,M)	0.2814	0.2885
(R)	0.2293	0.2969
(C,M)	**0.3650**	0.3413
(C)	0.2663	0.3093
(M)	0.2363	0.2835



(a) Including outliers

(b) Excluding outliers

Figure D.5: Feature Location effectiveness measures of optimal ($MRR = 0.3650$) and alternate ($MRR = 0.2601$) corpus configurations for Mahout v0.10.0



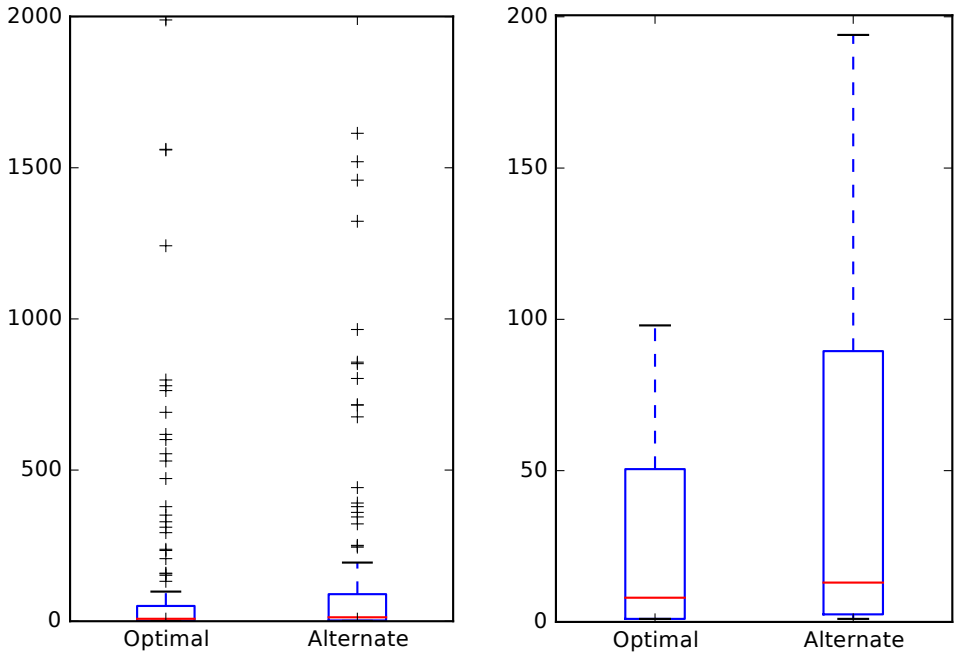
(a) Including outliers

(b) Excluding outliers

Figure D.6: Developer Identification effectiveness measures of optimal ($MRR = 0.3827$) and alternate ($MRR = 0.3413$) corpus configurations for Mahout v0.10.0

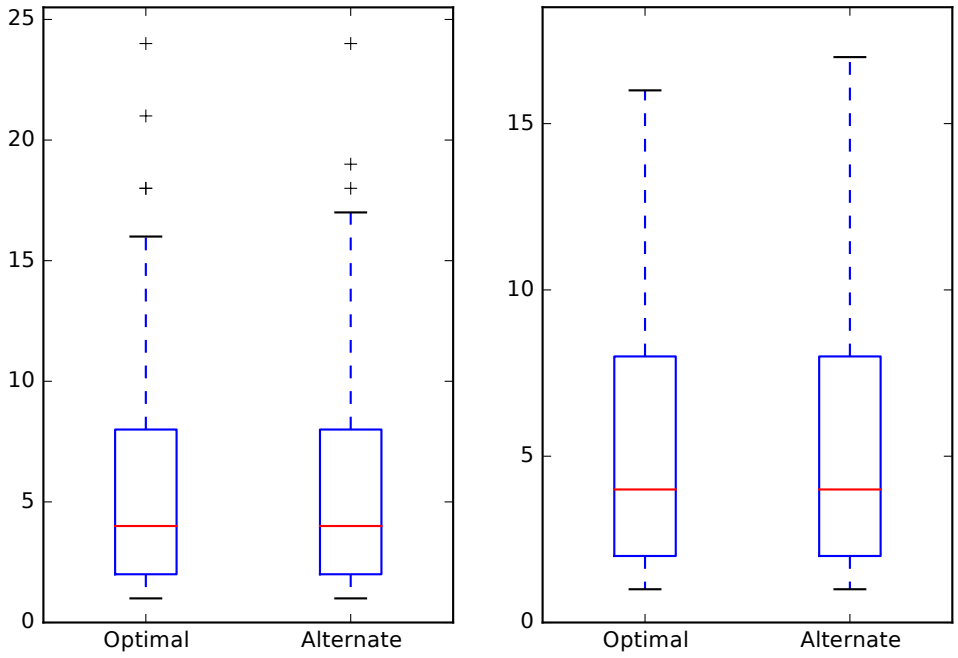
Table D.4: MRR values of OpenJPA v2.3.0 corpus construction sweep (*RQ 3.1* and *RQ 3.2*)

Configuration	FLT	DIT
(A,R,C,M)	0.3137	0.3833
(A,R,C)	0.2989	0.3648
(A,R,M)	0.2933	0.3916
(A,R)	0.2999	0.3568
(A,C,M)	**0.3687**	0.3935
(A,C)	0.2869	**0.4096**
(A,M)	0.3281	0.3349
(A)	0.2990	0.3446
(R,C,M)	0.2996	0.3544
(R,C)	0.2314	0.3256
(R,M)	0.2207	0.3317
(R)	0.2706	0.2563
(C,M)	0.2502	0.3236
(C)	0.2809	0.2744
(M)	0.2928	0.3157



(a) Including outliers (b) Excluding outliers

Figure D.7: Feature Location effectiveness measures of optimal ($MRR = 0.3687$) and alternate ($MRR = 0.2869$) corpus configurations for OpenJPA v2.3.0

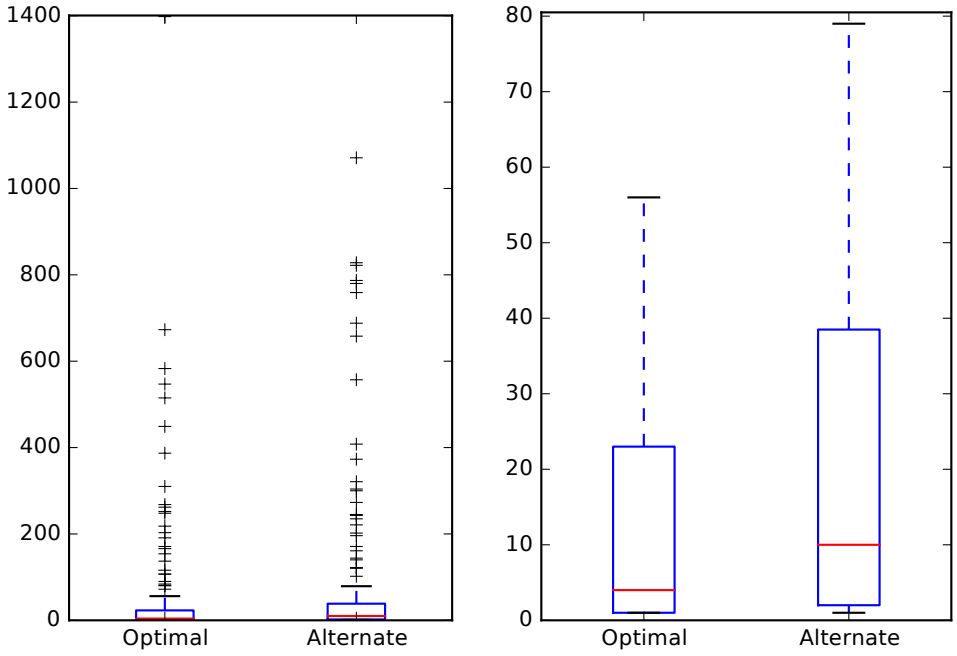


(a) Including outliers (b) Excluding outliers

Figure D.8: Developer Identification effectiveness measures of optimal ($MRR = 0.4096$) and alternate ($MRR = 0.3935$) corpus configurations for OpenJPA v2.3.0

Table D.5: MRR values of Pig v0.14.0 corpus construction sweep (*RQ 3.1* and *RQ 3.2*)

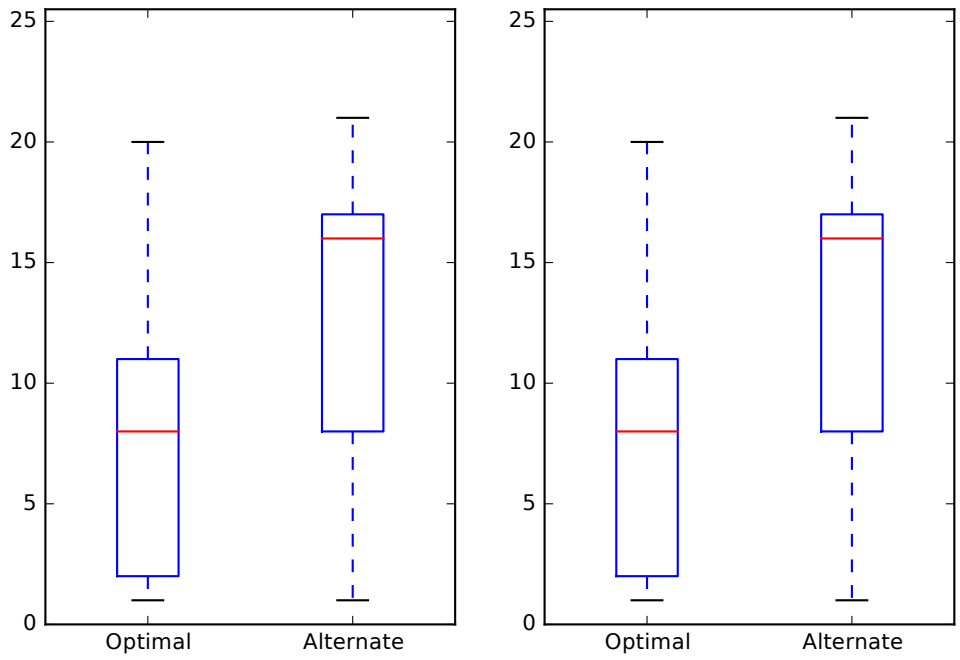
Configuration	FLT	DIT
(A,R,C,M)	0.3677	0.1801
(A,R,C)	0.3930	0.1759
(A,R,M)	0.3365	0.1824
(A,R)	**0.4360**	0.1437
(A,C,M)	0.4144	0.1874
(A,C)	0.3440	0.2032
(A,M)	0.3594	0.1416
(A)	0.3853	0.1421
(R,C,M)	0.3470	0.1771
(R,C)	0.2964	0.1316
(R,M)	0.3391	0.1694
(R)	0.2783	0.1533
(C,M)	0.3204	0.2933
(C)	0.3124	**0.3032**
(M)	0.3597	0.2703



(a) Including outliers

(b) Excluding outliers

Figure D.9: Feature Location effectiveness measures of optimal ($MRR = 0.4360$) and alternate ($MRR = 0.3124$) corpus configurations for Pig v0.14.0



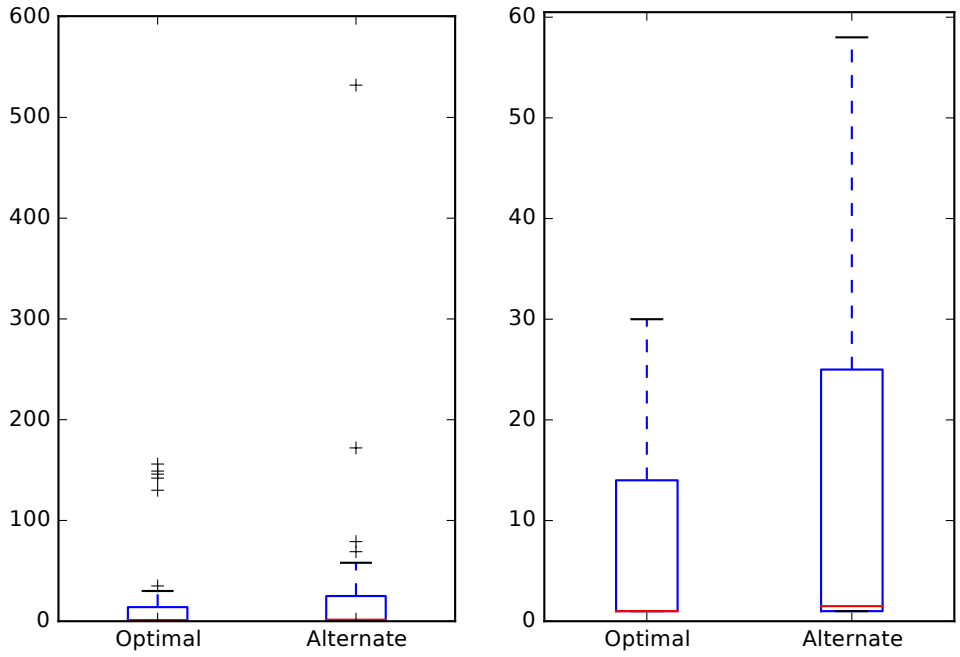
(a) Including outliers

(b) Excluding outliers

Figure D.10: Developer Identification effectiveness measures of optimal ($MRR = 0.3032$) and alternate ($MRR = 0.1437$) corpus configurations for Pig v0.14.0

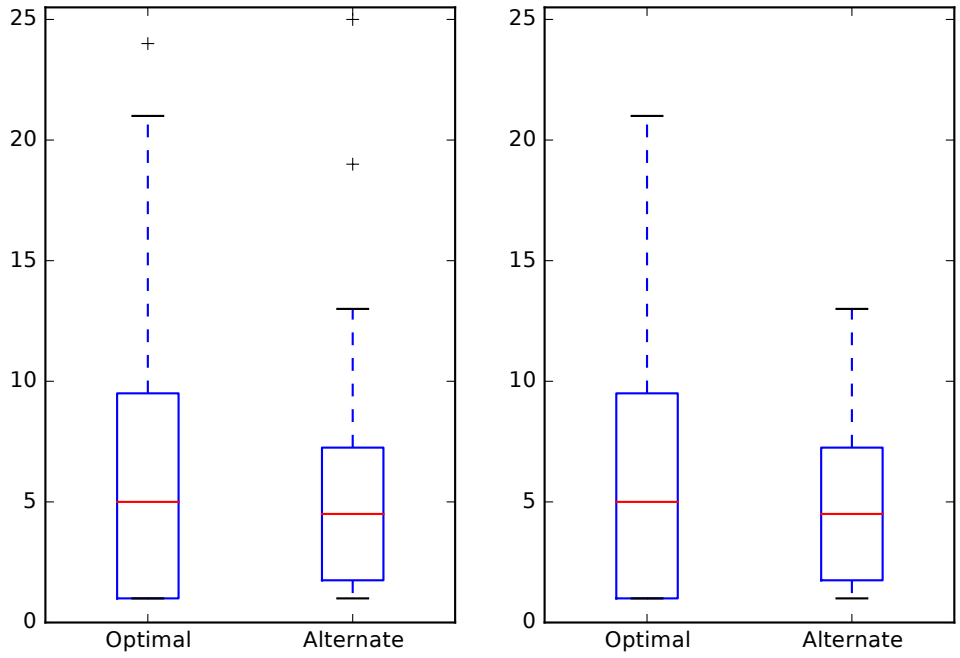
Table D.6: MRR values of Tika v1.8 corpus construction sweep (*RQ 3.1* and *RQ 3.2*)

Configuration	FLT	DIT
(A,R,C,M)	**0.6482**	0.3943
(A,R,C)	0.4033	0.3609
(A,R,M)	0.4713	0.3563
(A,R)	0.5231	0.4103
(A,C,M)	0.4759	0.4065
(A,C)	0.5572	**0.4522**
(A,M)	0.4864	0.3353
(A)	0.3829	0.4321
(R,C,M)	0.4494	0.3738
(R,C)	0.3940	0.3694
(R,M)	0.3759	0.3567
(R)	0.4109	0.3015
(C,M)	0.4507	0.3856
(C)	0.4198	0.2995
(M)	0.5475	0.3695



(a) Including outliers (b) Excluding outliers

Figure D.11: Feature Location effectiveness measures of optimal ($MRR = 0.6482$) and alternate ($MRR = 0.5572$) corpus configurations for Tika v1.8

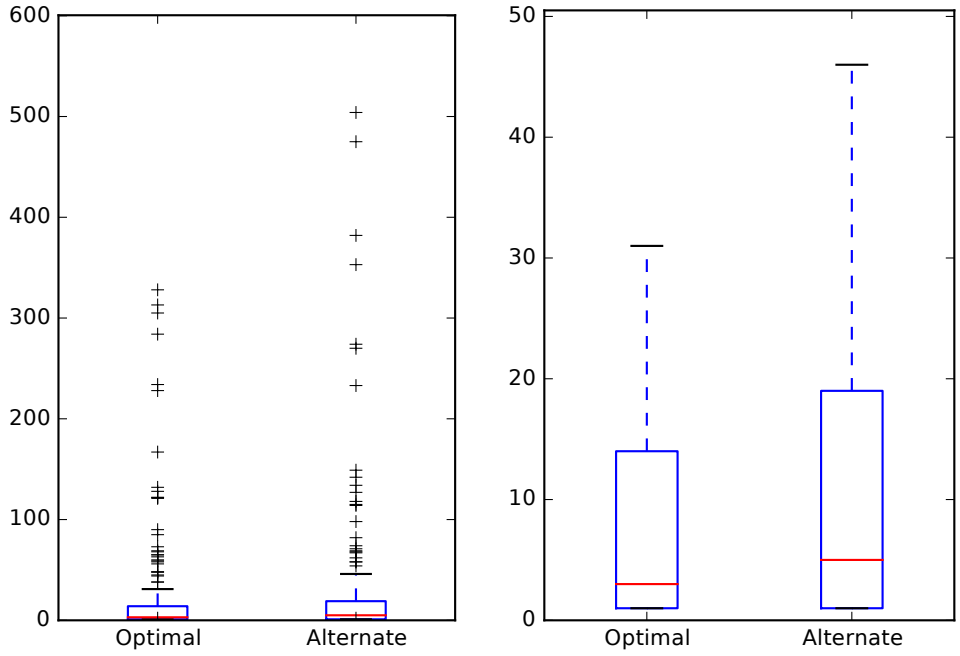


(a) Including outliers (b) Excluding outliers

Figure D.12: Developer Identification effectiveness measures of optimal ($MRR = 0.4522$) and alternate ($MRR = 0.3943$) corpus configurations for Tika v1.8

Table D.7: MRR values of ZooKeeper v3.5.0 corpus construction sweep (*RQ 3.1* and *RQ 3.2*)

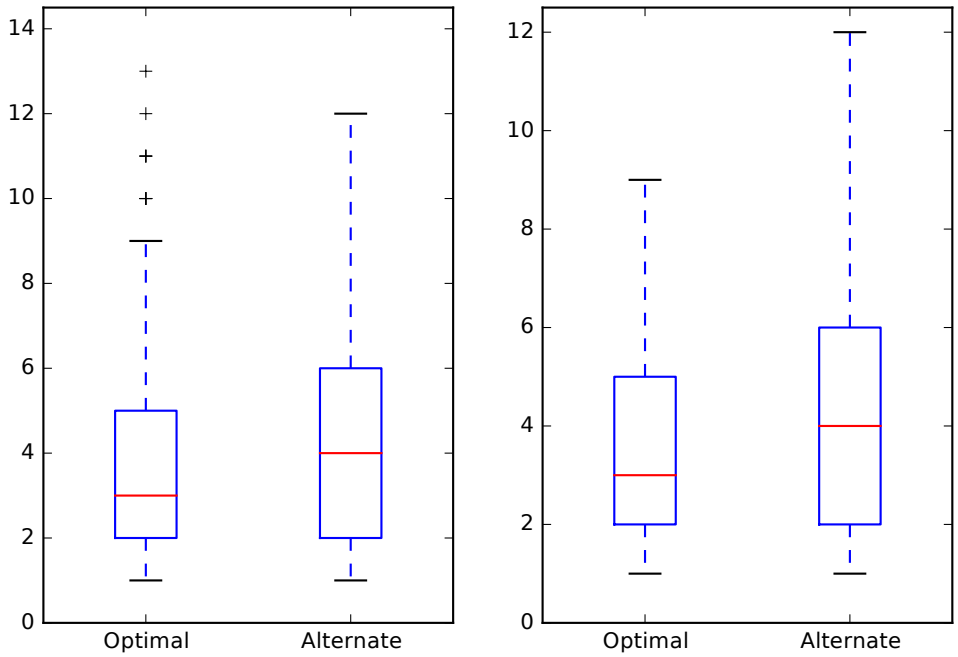
Configuration	FLT	DIT
(A,R,C,M)	0.4788	0.3776
(A,R,C)	0.4818	0.3985
(A,R,M)	0.4790	0.3706
(A,R)	0.4503	0.3858
(A,C,M)	**0.4950**	0.3667
(A,C)	0.4745	0.3579
(A,M)	0.4531	0.3643
(A)	0.4548	0.3637
(R,C,M)	0.4576	0.3189
(R,C)	0.4003	0.3456
(R,M)	0.4062	0.2970
(R)	0.3669	0.2973
(C,M)	0.4720	0.4247
(C)	0.3930	**0.4565**
(M)	0.4574	0.3188



(a) Including outliers

(b) Excluding outliers

Figure D.13: Feature Location effectiveness measures of optimal ($MRR = 0.4950$) and alternate ($MRR = 0.3930$) corpus configurations for ZooKeeper v3.5.0



(a) Including outliers

(b) Excluding outliers

Figure D.14: Developer Identification effectiveness measures of optimal ($MRR = 0.4565$) and alternate ($MRR = 0.3667$) corpus configurations for ZooKeeper v3.5.0

E. RP3: CORPUS CONSTRUCTION INCLUSION AND EXCLUSION

In these tables, we bold entries where the MRR is highest between each of the two inclusion or exclusion configurations.

Table E.1: Wilcoxon test results for additions inclusion and exclusion configurations of the FLT task for all subject systems (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(R,C,M)	**0.4315**	0.4013	$p < 0.01$	0.1366
(A,C,M)	(C,M)	**0.4517**	0.3971	$p < 0.01$	0.1273
(A,R,C)	(R,C)	**0.4092**	0.3443	$p < 0.01$	0.1739
(A,C)	(C)	**0.4010**	0.3386	0.1626	0.0637
(A,R,M)	(R,M)	**0.4119**	0.3488	$p < 0.01$	0.1825
(A,M)	(M)	**0.4147**	0.3838	0.8570	0.0082
(A,R)	(R)	**0.4214**	0.3151	$p < 0.01$	0.3554

Table E.2: Wilcoxon test results for additions inclusion and exclusion configurations of the DIT task for all subject systems (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(R,C,M)	**0.3784**	0.3391	$p < 0.01$	0.4174
(A,C,M)	(C,M)	0.3785	**0.4165**	$p < 0.01$	0.2580
(A,R,C)	(R,C)	**0.3770**	0.3373	$p < 0.01$	0.3923
(A,C)	(C)	0.3802	**0.4148**	$p < 0.01$	0.1686
(A,R,M)	(R,M)	**0.3646**	0.3308	$p < 0.01$	0.4145
(A,M)	(M)	**0.3537**	0.3359	0.1918	0.0498
(A,R)	(R)	**0.3462**	0.3147	$p < 0.01$	0.4047

Table E.3: Wilcoxon test results for removals inclusion and exclusion configurations of the FLT task for all subject systems (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,C,M)	0.4315	**0.4517**	0.2007	0.0620
(A,R,C)	(A,C)	**0.4092**	0.4010	0.4550	0.0356
(A,R,M)	(A,M)	0.4119	**0.4147**	0.0252	0.1053
(A,R)	(A)	**0.4214**	0.4031	0.2649	0.0528
(R,C,M)	(C,M)	**0.4013**	0.3971	0.3120	0.0467
(R,C)	(C)	**0.3443**	0.3386	0.1753	0.0612
(R,M)	(M)	0.3488	**0.3838**	$p < 0.01$	0.2400

Table E.4: Wilcoxon test results for removals inclusion and exclusion configurations of the DIT task for all subject systems (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,C,M)	0.3784	**0.3785**	$p < 0.01$	0.1405
(A,R,C)	(A,C)	0.3770	**0.3802**	$p < 0.01$	0.1162
(A,R,M)	(A,M)	**0.3646**	0.3537	$p < 0.01$	0.1847
(A,R)	(A)	**0.3462**	0.3380	0.0596	0.0779
(R,C,M)	(C,M)	0.3391	**0.4165**	$p < 0.01$	0.4621
(R,C)	(C)	0.3373	**0.4148**	$p < 0.01$	0.3971
(R,M)	(M)	0.3308	**0.3359**	$p < 0.01$	0.1625

Table E.5: Wilcoxon test results for context inclusion and exclusion configurations of the FLT task for all subject systems (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,R,M)	**0.4315**	0.4119	0.6584	0.0208
(A,C,M)	(A,M)	**0.4517**	0.4147	0.7929	0.0125
(A,R,C)	(A,R)	0.4092	**0.4214**	0.2354	0.0561
(A,C)	(A)	0.4010	**0.4031**	0.7706	0.0137
(R,C,M)	(R,M)	**0.4013**	0.3488	0.0637	0.0849
(C,M)	(M)	**0.3971**	0.3838	$p < 0.01$	0.1567
(R,C)	(R)	**0.3443**	0.3151	0.0104	0.1162

Table E.6: Wilcoxon test results for context inclusion and exclusion configurations of the DIT task for all subject systems (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,R,M)	**0.3784**	0.3646	$p < 0.01$	0.2445
(A,C,M)	(A,M)	**0.3785**	0.3537	$p < 0.01$	0.2631
(A,R,C)	(A,R)	**0.3770**	0.3462	$p < 0.01$	0.3748
(A,C)	(A)	**0.3802**	0.3380	$p < 0.01$	0.3302
(R,C,M)	(R,M)	**0.3391**	0.3308	$p < 0.01$	0.2567
(C,M)	(M)	**0.4165**	0.3359	$p < 0.01$	0.3783
(R,C)	(R)	**0.3373**	0.3147	$p < 0.01$	0.4001

Table E.7: Wilcoxon test results for message inclusion and exclusion configurations of the FLT task for all subject systems (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,R,C)	**0.4315**	0.4092	0.6327	0.0226
(A,C,M)	(A,C)	**0.4517**	0.4010	0.0161	0.1139
(A,R,M)	(A,R)	0.4119	**0.4214**	0.1471	0.0674
(A,M)	(A)	**0.4147**	0.4031	0.0109	0.1195
(R,C,M)	(R,C)	**0.4013**	0.3443	0.0310	0.0984
(C,M)	(C)	**0.3971**	0.3386	0.2897	0.0485
(R,M)	(R)	**0.3488**	0.3151	0.0387	0.0936

Table E.8: Wilcoxon test results for message inclusion and exclusion configurations of the DIT task for all subject systems (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,R,C)	**0.3784**	0.3770	0.0169	0.0995
(A,C,M)	(A,C)	0.3785	**0.3802**	0.1404	0.0616
(A,R,M)	(A,R)	**0.3646**	0.3462	$p < 0.01$	0.1454
(A,M)	(A)	**0.3537**	0.3380	0.5511	0.0241
(R,C,M)	(R,C)	**0.3391**	0.3373	0.3305	0.0399
(C,M)	(C)	**0.4165**	0.4148	0.6168	0.0204
(R,M)	(R)	**0.3308**	0.3147	$p < 0.01$	0.1878

Table E.9: Wilcoxon test results for additions inclusion and exclusion configurations of the FLT task for BookKeeper v4.3.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(R,C,M)	**0.5246**	0.4786	0.1602	0.1586
(A,C,M)	(C,M)	**0.5327**	0.4968	0.1358	0.1623
(A,R,C)	(R,C)	**0.4567**	0.4036	0.0398	0.2208
(A,C)	(C)	**0.4450**	0.3366	$p < 0.01$	0.3497
(A,R,M)	(R,M)	**0.5316**	0.3978	$p < 0.01$	0.4635
(A,M)	(M)	**0.5260**	0.3826	0.1016	0.1744
(A,R)	(R)	**0.4970**	0.3195	$p < 0.01$	0.6769

Table E.10: Wilcoxon test results for additions inclusion and exclusion configurations of the DIT task for BookKeeper v4.3.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(R,C,M)	**0.7216**	0.6530	$p < 0.01$	0.5688
(A,C,M)	(C,M)	0.6974	**0.7114**	$p < 0.01$	0.4427
(A,R,C)	(R,C)	0.6513	**0.6526**	0.4967	0.0817
(A,C)	(C)	0.6517	**0.7057**	$p < 0.01$	0.5178
(A,R,M)	(R,M)	0.6465	**0.6507**	0.3267	0.1129
(A,M)	(M)	**0.6722**	0.5132	0.0172	0.2336
(A,R)	(R)	0.5425	**0.6375**	$p < 0.01$	0.4316

Table E.11: Wilcoxon test results for removals inclusion and exclusion configurations of the FLT task for BookKeeper v4.3.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,C,M)	0.5246	**0.5327**	0.7311	0.0410
(A,R,C)	(A,C)	**0.4567**	0.4450	0.4925	0.0796
(A,R,M)	(A,M)	**0.5316**	0.5260	0.8400	0.0238
(A,R)	(A)	**0.4970**	0.4881	0.9703	0.0045
(R,C,M)	(C,M)	0.4786	**0.4968**	0.8254	0.0251
(R,C)	(C)	**0.4036**	0.3366	0.4352	0.0818
(R,M)	(M)	**0.3978**	0.3826	$p < 0.01$	0.3029

Table E.12: Wilcoxon test results for removals inclusion and exclusion configurations of the DIT task for BookKeeper v4.3.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,C,M)	**0.7216**	0.6974	$p < 0.01$	0.7783
(A,R,C)	(A,C)	0.6513	**0.6517**	0.9876	0.0038
(A,R,M)	(A,M)	0.6465	**0.6722**	0.1248	0.2500
(A,R)	(A)	**0.5425**	0.4821	0.3253	0.1050
(R,C,M)	(C,M)	0.6530	**0.7114**	$p < 0.01$	0.4669
(R,C)	(C)	0.6526	**0.7057**	$p < 0.01$	0.5674
(R,M)	(M)	**0.6507**	0.5132	0.1790	0.1316

Table E.13: Wilcoxon test results for context inclusion and exclusion configurations of the FLT task for BookKeeper v4.3.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,R,M)	0.5246	**0.5316**	0.7160	0.0422
(A,C,M)	(A,M)	**0.5327**	0.5260	0.7610	0.0354
(A,R,C)	(A,R)	0.4567	**0.4970**	0.3459	0.1060
(A,C)	(A)	0.4450	**0.4881**	0.3057	0.1163
(R,C,M)	(R,M)	**0.4786**	0.3978	0.0503	0.2151
(C,M)	(M)	**0.4968**	0.3826	0.3570	0.0973
(R,C)	(R)	**0.4036**	0.3195	$p < 0.01$	0.3855

Table E.14: Wilcoxon test results for context inclusion and exclusion configurations of the DIT task for BookKeeper v4.3.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,R,M)	**0.7216**	0.6465	$p < 0.01$	0.8575
(A,C,M)	(A,M)	**0.6974**	0.6722	0.1609	0.2667
(A,R,C)	(A,R)	**0.6513**	0.5425	$p < 0.01$	0.6881
(A,C)	(A)	**0.6517**	0.4821	$p < 0.01$	0.5113
(R,C,M)	(R,M)	**0.6530**	0.6507	0.6651	0.0646
(C,M)	(M)	**0.7114**	0.5132	$p < 0.01$	0.6510
(R,C)	(R)	**0.6526**	0.6375	0.4704	0.0966

Table E.15: Wilcoxon test results for message inclusion and exclusion configurations of the FLT task for BookKeeper v4.3.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,R,C)	**0.5246**	0.4567	0.0350	0.2394
(A,C,M)	(A,C)	**0.5327**	0.4450	0.0981	0.1860
(A,R,M)	(A,R)	**0.5316**	0.4970	0.1174	0.1759
(A,M)	(A)	**0.5260**	0.4881	0.1579	0.1634
(R,C,M)	(R,C)	**0.4786**	0.4036	0.0883	0.1840
(C,M)	(C)	**0.4968**	0.3366	0.0426	0.2160
(R,M)	(R)	**0.3978**	0.3195	$p < 0.01$	0.4133

Table E.16: Wilcoxon test results for message inclusion and exclusion configurations of the DIT task for BookKeeper v4.3.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,R,C)	**0.7216**	0.6513	$p < 0.01$	0.8459
(A,C,M)	(A,C)	**0.6974**	0.6517	0.0107	0.3778
(A,R,M)	(A,R)	**0.6465**	0.5425	$p < 0.01$	0.7143
(A,M)	(A)	**0.6722**	0.4821	$p < 0.01$	0.6254
(R,C,M)	(R,C)	**0.6530**	0.6526	0.5719	0.0909
(C,M)	(C)	**0.7114**	0.7057	0.5630	0.0824
(R,M)	(R)	**0.6507**	0.6375	0.5727	0.0768

Table E.17: Wilcoxon test results for additions inclusion and exclusion configurations of the FLT task for Mahout v0.10.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(R,C,M)	0.3119	**0.3305**	0.1246	0.2764
(A,C,M)	(C,M)	0.3422	**0.3650**	0.1541	0.2598
(A,R,C)	(R,C)	0.2730	**0.3325**	0.5359	0.1081
(A,C)	(C)	**0.3063**	0.2663	0.4180	0.1365
(A,R,M)	(R,M)	0.2766	**0.2814**	0.5053	0.1150
(A,M)	(M)	**0.2799**	0.2363	0.6463	0.0814
(A,R)	(R)	**0.2606**	0.2293	0.1929	0.2190

Table E.18: Wilcoxon test results for additions inclusion and exclusion configurations of the DIT task for Mahout v0.10.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(R,C,M)	**0.2785**	0.2505	$p < 0.01$	0.3554
(A,C,M)	(C,M)	0.3124	**0.3413**	0.2607	0.1233
(A,R,C)	(R,C)	**0.3340**	0.2719	0.0151	0.2686
(A,C)	(C)	**0.3492**	0.3093	0.2643	0.1178
(A,R,M)	(R,M)	0.2797	**0.2885**	0.1351	0.1613
(A,M)	(M)	**0.3116**	0.2835	0.9394	0.0081
(A,R)	(R)	**0.3052**	0.2969	0.0228	0.2534

Table E.19: Wilcoxon test results for removals inclusion and exclusion configurations of the FLT task for Mahout v0.10.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,C,M)	0.3119	**0.3422**	0.7029	0.0687
(A,R,C)	(A,C)	0.2730	**0.3063**	0.5824	0.0973
(A,R,M)	(A,M)	0.2766	**0.2799**	0.2817	0.1832
(A,R)	(A)	**0.2606**	0.2601	0.9253	0.0177
(R,C,M)	(C,M)	0.3305	**0.3650**	0.4277	0.1451
(R,C)	(C)	**0.3325**	0.2663	0.2934	0.1828
(R,M)	(M)	**0.2814**	0.2363	0.2639	0.1966

Table E.20: Wilcoxon test results for removals inclusion and exclusion configurations of the DIT task for Mahout v0.10.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,C,M)	0.2785	**0.3124**	0.2928	0.1214
(A,R,C)	(A,C)	0.3340	**0.3492**	0.9509	0.0073
(A,R,M)	(A,M)	0.2797	**0.3116**	0.1800	0.1443
(A,R)	(A)	0.3052	**0.3827**	0.0598	0.2108
(R,C,M)	(C,M)	0.2505	**0.3413**	$p < 0.01$	0.3051
(R,C)	(C)	0.2719	**0.3093**	0.9174	0.0111
(R,M)	(M)	**0.2885**	0.2835	0.0835	0.1813

Table E.21: Wilcoxon test results for context inclusion and exclusion configurations of the FLT task for Mahout v0.10.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,R,M)	**0.3119**	0.2766	0.6885	0.0696
(A,C,M)	(A,M)	**0.3422**	0.2799	0.8091	0.0433
(A,R,C)	(A,R)	**0.2730**	0.2606	0.5457	0.1043
(A,C)	(A)	**0.3063**	0.2601	0.9550	0.0106
(R,C,M)	(R,M)	**0.3305**	0.2814	0.7356	0.0609
(C,M)	(M)	**0.3650**	0.2363	0.1221	0.2717
(R,C)	(R)	**0.3325**	0.2293	0.0443	0.3531

Table E.22: Wilcoxon test results for context inclusion and exclusion configurations of the DIT task for Mahout v0.10.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,R,M)	0.2785	**0.2797**	0.1708	0.1475
(A,C,M)	(A,M)	**0.3124**	0.3116	0.2858	0.1200
(A,R,C)	(A,R)	**0.3340**	0.3052	$p < 0.01$	0.3107
(A,C)	(A)	0.3492	**0.3827**	0.8116	0.0258
(R,C,M)	(R,M)	0.2505	**0.2885**	0.8384	0.0221
(C,M)	(M)	**0.3413**	0.2835	0.6532	0.0469
(R,C)	(R)	0.2719	**0.2969**	0.0166	0.2607

Table E.23: Wilcoxon test results for message inclusion and exclusion configurations of the FLT task for Mahout v0.10.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,R,C)	**0.3119**	0.2730	0.2088	0.2237
(A,C,M)	(A,C)	**0.3422**	0.3063	0.5500	0.1057
(A,R,M)	(A,R)	**0.2766**	0.2606	0.6434	0.0802
(A,M)	(A)	**0.2799**	0.2601	0.6594	0.0782
(R,C,M)	(R,C)	0.3305	**0.3325**	0.9357	0.0159
(C,M)	(C)	**0.3650**	0.2663	0.3501	0.1626
(R,M)	(R)	**0.2814**	0.2293	0.1905	0.2358

Table E.24: Wilcoxon test results for message inclusion and exclusion configurations of the DIT task for Mahout v0.10.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,R,C)	0.2785	**0.3340**	0.0514	0.2162
(A,C,M)	(A,C)	0.3124	**0.3492**	0.7202	0.0390
(A,R,M)	(A,R)	0.2797	**0.3052**	0.3502	0.1020
(A,M)	(A)	0.3116	**0.3827**	0.2043	0.1418
(R,C,M)	(R,C)	0.2505	**0.2719**	0.0382	0.2169
(C,M)	(C)	**0.3413**	0.3093	0.5200	0.0694
(R,M)	(R)	0.2885	**0.2969**	0.1157	0.1696

Table E.25: Wilcoxon test results for additions inclusion and exclusion configurations of the FLT task for OpenJPA v2.3.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(R,C,M)	**0.3137**	0.2996	0.5591	0.0640
(A,C,M)	(C,M)	**0.3687**	0.2502	0.3994	0.0911
(A,R,C)	(R,C)	**0.2989**	0.2314	0.1271	0.1655
(A,C)	(C)	**0.2869**	0.2809	0.5079	0.0726
(A,R,M)	(R,M)	**0.2933**	0.2207	0.0785	0.1892
(A,M)	(M)	**0.3281**	0.2928	0.7708	0.0316
(A,R)	(R)	**0.2999**	0.2706	0.1349	0.1600

Table E.26: Wilcoxon test results for additions inclusion and exclusion configurations of the DIT task for OpenJPA v2.3.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(R,C,M)	**0.3833**	0.3544	$p < 0.01$	0.3268
(A,C,M)	(C,M)	**0.3935**	0.3236	0.2093	0.1386
(A,R,C)	(R,C)	**0.3648**	0.3256	0.1138	0.1676
(A,C)	(C)	**0.4096**	0.2744	$p < 0.01$	0.3940
(A,R,M)	(R,M)	**0.3916**	0.3317	$p < 0.01$	0.3639
(A,M)	(M)	**0.3349**	0.3157	0.0357	0.2282
(A,R)	(R)	**0.3568**	0.2563	$p < 0.01$	0.5402

Table E.27: Wilcoxon test results for removals inclusion and exclusion configurations of the FLT task for OpenJPA v2.3.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,C,M)	0.3137	**0.3687**	0.0275	0.2480
(A,R,C)	(A,C)	**0.2989**	0.2869	0.2604	0.1273
(A,R,M)	(A,M)	0.2933	**0.3281**	0.1028	0.1794
(A,R)	(A)	**0.2999**	0.2990	0.1323	0.1654
(R,C,M)	(C,M)	**0.2996**	0.2502	0.6492	0.0501
(R,C)	(C)	0.2314	**0.2809**	0.2256	0.1292
(R,M)	(M)	0.2207	**0.2928**	$p < 0.01$	0.3409

Table E.28: Wilcoxon test results for removals inclusion and exclusion configurations of the DIT task for OpenJPA v2.3.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,C,M)	0.3833	**0.3935**	0.6557	0.0488
(A,R,C)	(A,C)	0.3648	**0.4096**	0.0961	0.1821
(A,R,M)	(A,M)	**0.3916**	0.3349	$p < 0.01$	0.3777
(A,R)	(A)	**0.3568**	0.3446	$p < 0.01$	0.4016
(R,C,M)	(C,M)	**0.3544**	0.3236	0.1048	0.1760
(R,C)	(C)	**0.3256**	0.2744	0.0224	0.2418
(R,M)	(M)	**0.3317**	0.3157	0.0531	0.2020

Table E.29: Wilcoxon test results for context inclusion and exclusion configurations of the FLT task for OpenJPA v2.3.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,R,M)	**0.3137**	0.2933	0.6027	0.0573
(A,C,M)	(A,M)	**0.3687**	0.3281	0.7511	0.0364
(A,R,C)	(A,R)	0.2989	**0.2999**	0.9225	0.0107
(A,C)	(A)	0.2869	**0.2990**	0.6567	0.0492
(R,C,M)	(R,M)	**0.2996**	0.2207	0.0642	0.1990
(C,M)	(M)	0.2502	**0.2928**	$p < 0.01$	0.3195
(R,C)	(R)	0.2314	**0.2706**	0.4836	0.0761

Table E.30: Wilcoxon test results for context inclusion and exclusion configurations of the DIT task for OpenJPA v2.3.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,R,M)	0.3833	**0.3916**	0.3609	0.1031
(A,C,M)	(A,M)	**0.3935**	0.3349	$p < 0.01$	0.2939
(A,R,C)	(A,R)	**0.3648**	0.3568	0.4329	0.0857
(A,C)	(A)	**0.4096**	0.3446	$p < 0.01$	0.3531
(R,C,M)	(R,M)	**0.3544**	0.3317	0.9438	0.0082
(C,M)	(M)	**0.3236**	0.3157	$p < 0.01$	0.3753
(R,C)	(R)	**0.3256**	0.2563	$p < 0.01$	0.4766

Table E.31: Wilcoxon test results for message inclusion and exclusion configurations of the FLT task for OpenJPA v2.3.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,R,C)	**0.3137**	0.2989	0.1274	0.1668
(A,C,M)	(A,C)	**0.3687**	0.2869	0.1908	0.1425
(A,R,M)	(A,R)	0.2933	**0.2999**	0.2438	0.1288
(A,M)	(A)	**0.3281**	0.2990	0.0246	0.2438
(R,C,M)	(R,C)	**0.2996**	0.2314	0.6451	0.0496
(C,M)	(C)	0.2502	**0.2809**	0.4683	0.0784
(R,M)	(R)	0.2207	**0.2706**	$p < 0.01$	0.3616

Table E.32: Wilcoxon test results for message inclusion and exclusion configurations of the DIT task for OpenJPA v2.3.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,R,C)	**0.3833**	0.3648	0.5215	0.0690
(A,C,M)	(A,C)	0.3935	**0.4096**	0.5454	0.0687
(A,R,M)	(A,R)	**0.3916**	0.3568	0.4277	0.0890
(A,M)	(A)	0.3349	**0.3446**	0.5640	0.0619
(R,C,M)	(R,C)	**0.3544**	0.3256	0.6990	0.0414
(C,M)	(C)	**0.3236**	0.2744	$p < 0.01$	0.4130
(R,M)	(R)	**0.3317**	0.2563	$p < 0.01$	0.5261

Table E.33: Wilcoxon test results for additions inclusion and exclusion configurations of the FLT task for Pig v0.14.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(R,C,M)	**0.3677**	0.3470	0.0775	0.1667
(A,C,M)	(C,M)	**0.4144**	0.3204	$p < 0.01$	0.3007
(A,R,C)	(R,C)	**0.3930**	0.2964	0.3012	0.0984
(A,C)	(C)	**0.3440**	0.3124	0.7376	0.0316
(A,R,M)	(R,M)	0.3365	**0.3391**	0.3280	0.0924
(A,M)	(M)	0.3594	**0.3597**	0.6056	0.0483
(A,R)	(R)	**0.4360**	0.2783	$p < 0.01$	0.2698

Table E.34: Wilcoxon test results for additions inclusion and exclusion configurations of the DIT task for Pig v0.14.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(R,C,M)	**0.1801**	0.1771	$p < 0.01$	0.3317
(A,C,M)	(C,M)	0.1874	**0.2933**	$p < 0.01$	0.6814
(A,R,C)	(R,C)	**0.1759**	0.1316	$p < 0.01$	0.7057
(A,C)	(C)	0.2032	**0.3032**	$p < 0.01$	0.4478
(A,R,M)	(R,M)	**0.1824**	0.1694	$p < 0.01$	0.5469
(A,M)	(M)	0.1416	**0.2703**	$p < 0.01$	0.2574
(A,R)	(R)	0.1437	**0.1533**	$p < 0.01$	0.3251

Table E.35: Wilcoxon test results for removals inclusion and exclusion configurations of the FLT task for Pig v0.14.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,C,M)	0.3677	**0.4144**	0.5091	0.0649
(A,R,C)	(A,C)	**0.3930**	0.3440	0.6846	0.0391
(A,R,M)	(A,M)	0.3365	**0.3594**	0.1790	0.1296
(A,R)	(A)	**0.4360**	0.3853	0.9867	0.0017
(R,C,M)	(C,M)	**0.3470**	0.3204	0.9496	0.0061
(R,C)	(C)	0.2964	**0.3124**	0.8455	0.0181
(R,M)	(M)	0.3391	**0.3597**	0.1376	0.1389

Table E.36: Wilcoxon test results for removals inclusion and exclusion configurations of the DIT task for Pig v0.14.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,C,M)	0.1801	**0.1874**	$p < 0.01$	0.2497
(A,R,C)	(A,C)	0.1759	**0.2032**	0.8779	0.0129
(A,R,M)	(A,M)	**0.1824**	0.1416	$p < 0.01$	0.6179
(A,R)	(A)	**0.1437**	0.1421	0.8170	0.0198
(R,C,M)	(C,M)	0.1771	**0.2933**	$p < 0.01$	0.7480
(R,C)	(C)	0.1316	**0.3032**	$p < 0.01$	0.8995
(R,M)	(M)	0.1694	**0.2703**	$p < 0.01$	0.3320

Table E.37: Wilcoxon test results for context inclusion and exclusion configurations of the FLT task for Pig v0.14.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,R,M)	**0.3677**	0.3365	0.2923	0.0999
(A,C,M)	(A,M)	**0.4144**	0.3594	0.5846	0.0531
(A,R,C)	(A,R)	0.3930	**0.4360**	0.0732	0.1825
(A,C)	(A)	0.3440	**0.3853**	0.1779	0.1290
(R,C,M)	(R,M)	**0.3470**	0.3391	0.9068	0.0112
(C,M)	(M)	0.3204	**0.3597**	0.0130	0.2378
(R,C)	(R)	**0.2964**	0.2783	0.9077	0.0108

Table E.38: Wilcoxon test results for context inclusion and exclusion configurations of the DIT task for Pig v0.14.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,R,M)	0.1801	**0.1824**	$p < 0.01$	0.3641
(A,C,M)	(A,M)	**0.1874**	0.1416	$p < 0.01$	0.5968
(A,R,C)	(A,R)	**0.1759**	0.1437	$p < 0.01$	0.7858
(A,C)	(A)	**0.2032**	0.1421	$p < 0.01$	0.8120
(R,C,M)	(R,M)	**0.1771**	0.1694	$p < 0.01$	0.5928
(C,M)	(M)	**0.2933**	0.2703	$p < 0.01$	0.5915
(R,C)	(R)	0.1316	**0.1533**	$p < 0.01$	0.4218

Table E.39: Wilcoxon test results for message inclusion and exclusion configurations of the FLT task for Pig v0.14.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,R,C)	0.3677	**0.3930**	0.4915	0.0673
(A,C,M)	(A,C)	**0.4144**	0.3440	0.2285	0.1161
(A,R,M)	(A,R)	0.3365	**0.4360**	$p < 0.01$	0.2882
(A,M)	(A)	0.3594	**0.3853**	0.7708	0.0281
(R,C,M)	(R,C)	**0.3470**	0.2964	0.2954	0.0985
(C,M)	(C)	**0.3204**	0.3124	0.4229	0.0755
(R,M)	(R)	**0.3391**	0.2783	0.4010	0.0786

Table E.40: Wilcoxon test results for message inclusion and exclusion configurations of the DIT task for Pig v0.14.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,R,C)	**0.1801**	0.1759	0.0107	0.2154
(A,C,M)	(A,C)	0.1874	**0.2032**	$p < 0.01$	0.3628
(A,R,M)	(A,R)	**0.1824**	0.1437	$p < 0.01$	0.7016
(A,M)	(A)	0.1416	**0.1421**	0.7047	0.0317
(R,C,M)	(R,C)	**0.1771**	0.1316	$p < 0.01$	0.4832
(C,M)	(C)	0.2933	**0.3032**	0.7497	0.0276
(R,M)	(R)	**0.1694**	0.1533	$p < 0.01$	0.2232

Table E.41: Wilcoxon test results for additions inclusion and exclusion configurations of the FLT task for Tika v1.8 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(R,C,M)	**0.6482**	0.4494	$p < 0.01$	0.6333
(A,C,M)	(C,M)	**0.4759**	0.4507	0.3986	0.1847
(A,R,C)	(R,C)	**0.4033**	0.3940	0.3181	0.2108
(A,C)	(C)	**0.5572**	0.4198	0.7409	0.0739
(A,R,M)	(R,M)	**0.4713**	0.3759	0.2378	0.2529
(A,M)	(M)	0.4864	**0.5475**	0.1710	0.3138
(A,R)	(R)	**0.5231**	0.4109	0.0316	0.4495

Table E.42: Wilcoxon test results for additions inclusion and exclusion configurations of the DIT task for Tika v1.8 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(R,C,M)	**0.3943**	0.3738	0.7728	0.0605
(A,C,M)	(C,M)	**0.4065**	0.3856	0.6123	0.1008
(A,R,C)	(R,C)	0.3609	**0.3694**	0.8440	0.0430
(A,C)	(C)	**0.4522**	0.2995	0.9408	0.0159
(A,R,M)	(R,M)	0.3563	**0.3567**	0.9177	0.0227
(A,M)	(M)	0.3353	**0.3695**	0.0873	0.3273
(A,R)	(R)	**0.4103**	0.3015	0.0218	0.4504

Table E.43: Wilcoxon test results for removals inclusion and exclusion configurations of the FLT task for Tika v1.8 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,C,M)	**0.6482**	0.4759	0.3451	0.2233
(A,R,C)	(A,C)	0.4033	**0.5572**	0.2587	0.2463
(A,R,M)	(A,M)	0.4713	**0.4864**	0.7454	0.0830
(A,R)	(A)	**0.5231**	0.3829	0.0105	0.5542
(R,C,M)	(C,M)	0.4494	**0.4507**	0.3328	0.2118
(R,C)	(C)	0.3940	**0.4198**	0.3622	0.1995
(R,M)	(M)	0.3759	**0.5475**	$p < 0.01$	0.7238

Table E.44: Wilcoxon test results for removals inclusion and exclusion configurations of the DIT task for Tika v1.8 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,C,M)	0.3943	**0.4065**	0.9517	0.0159
(A,R,C)	(A,C)	0.3609	**0.4522**	0.5868	0.1217
(A,R,M)	(A,M)	**0.3563**	0.3353	0.6781	0.0838
(A,R)	(A)	0.4103	**0.4321**	0.8883	0.0342
(R,C,M)	(C,M)	0.3738	**0.3856**	0.2670	0.2254
(R,C)	(C)	**0.3694**	0.2995	0.6496	0.0968
(R,M)	(M)	0.3567	**0.3695**	0.1038	0.3108

Table E.45: Wilcoxon test results for context inclusion and exclusion configurations of the FLT task for Tika v1.8 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,R,M)	**0.6482**	0.4713	0.2467	0.2733
(A,C,M)	(A,M)	0.4759	**0.4864**	0.6289	0.1111
(A,R,C)	(A,R)	0.4033	**0.5231**	0.2050	0.2759
(A,C)	(A)	**0.5572**	0.3829	0.0253	0.5043
(R,C,M)	(R,M)	**0.4494**	0.3759	0.5160	0.1376
(C,M)	(M)	0.4507	**0.5475**	0.1674	0.3005
(R,C)	(R)	0.3940	**0.4109**	0.8286	0.0493

Table E.46: Wilcoxon test results for context inclusion and exclusion configurations of the DIT task for Tika v1.8 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,R,M)	**0.3943**	0.3563	0.0181	0.5099
(A,C,M)	(A,M)	**0.4065**	0.3353	0.1080	0.3425
(A,R,C)	(A,R)	0.3609	**0.4103**	0.0786	0.3862
(A,C)	(A)	**0.4522**	0.4321	0.3778	0.1958
(R,C,M)	(R,M)	**0.3738**	0.3567	0.0950	0.3333
(C,M)	(M)	**0.3856**	0.3695	0.8202	0.0484
(R,C)	(R)	**0.3694**	0.3015	0.0873	0.3528

Table E.47: Wilcoxon test results for message inclusion and exclusion configurations of the FLT task for Tika v1.8 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,R,C)	**0.6482**	0.4033	0.0485	0.4207
(A,C,M)	(A,C)	0.4759	**0.5572**	0.9493	0.0171
(A,R,M)	(A,R)	0.4713	**0.5231**	0.6567	0.1046
(A,M)	(A)	**0.4864**	0.3829	0.0527	0.4138
(R,C,M)	(R,C)	**0.4494**	0.3940	0.2741	0.2389
(C,M)	(C)	**0.4507**	0.4198	0.4160	0.1852
(R,M)	(R)	0.3759	**0.4109**	0.5801	0.1243

Table E.48: Wilcoxon test results for message inclusion and exclusion configurations of the DIT task for Tika v1.8 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,R,C)	**0.3943**	0.3609	0.2319	0.2460
(A,C,M)	(A,C)	0.4065	**0.4522**	0.6175	0.1080
(A,R,M)	(A,R)	0.3563	**0.4103**	0.0399	0.4082
(A,M)	(A)	0.3353	**0.4321**	0.0949	0.3540
(R,C,M)	(R,C)	**0.3738**	0.3694	0.6029	0.1061
(C,M)	(C)	**0.3856**	0.2995	0.1316	0.3049
(R,M)	(R)	**0.3567**	0.3015	0.5962	0.1070

Table E.49: Wilcoxon test results for additions inclusion and exclusion configurations of the FLT task for ZooKeeper v3.5.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(R,C,M)	**0.4788**	0.4576	0.0180	0.1989
(A,C,M)	(C,M)	**0.4950**	0.4720	0.5605	0.0505
(A,R,C)	(R,C)	**0.4818**	0.4003	$p < 0.01$	0.3071
(A,C)	(C)	**0.4745**	0.3930	0.6584	0.0378
(A,R,M)	(R,M)	**0.4790**	0.4062	0.0729	0.1508
(A,M)	(M)	0.4531	**0.4574**	0.6427	0.0385
(A,R)	(R)	**0.4503**	0.3669	$p < 0.01$	0.3557

Table E.50: Wilcoxon test results for additions inclusion and exclusion configurations of the DIT task for ZooKeeper v3.5.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(R,C,M)	**0.3776**	0.3189	$p < 0.01$	0.5378
(A,C,M)	(C,M)	0.3667	**0.4247**	$p < 0.01$	0.2411
(A,R,C)	(R,C)	**0.3985**	0.3456	$p < 0.01$	0.3722
(A,C)	(C)	0.3579	**0.4565**	$p < 0.01$	0.3736
(A,R,M)	(R,M)	**0.3706**	0.2970	$p < 0.01$	0.5231
(A,M)	(M)	**0.3643**	0.3188	$p < 0.01$	0.2225
(A,R)	(R)	**0.3858**	0.2973	$p < 0.01$	0.6427

Table E.51: Wilcoxon test results for removals inclusion and exclusion configurations of the FLT task for ZooKeeper v3.5.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,C,M)	0.4788	**0.4950**	0.5259	0.0572
(A,R,C)	(A,C)	**0.4818**	0.4745	0.5493	0.0531
(A,R,M)	(A,M)	**0.4790**	0.4531	0.5439	0.0516
(A,R)	(A)	0.4503	**0.4548**	0.6461	0.0401
(R,C,M)	(C,M)	0.4576	**0.4720**	0.3425	0.0787
(R,C)	(C)	**0.4003**	0.3930	0.0210	0.1935
(R,M)	(M)	0.4062	**0.4574**	$p < 0.01$	0.2778

Table E.52: Wilcoxon test results for removals inclusion and exclusion configurations of the DIT task for ZooKeeper v3.5.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,C,M)	**0.3776**	0.3667	$p < 0.01$	0.2277
(A,R,C)	(A,C)	**0.3985**	0.3579	$p < 0.01$	0.3952
(A,R,M)	(A,M)	**0.3706**	0.3643	0.9356	0.0059
(A,R)	(A)	**0.3858**	0.3637	$p < 0.01$	0.2080
(R,C,M)	(C,M)	0.3189	**0.4247**	$p < 0.01$	0.4575
(R,C)	(C)	0.3456	**0.4565**	$p < 0.01$	0.4250
(R,M)	(M)	0.2970	**0.3188**	$p < 0.01$	0.2718

Table E.53: Wilcoxon test results for context inclusion and exclusion configurations of the FLT task for ZooKeeper v3.5.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,R,M)	0.4788	**0.4790**	0.3947	0.0738
(A,C,M)	(A,M)	**0.4950**	0.4531	0.3614	0.0800
(A,R,C)	(A,R)	**0.4818**	0.4503	0.2724	0.0948
(A,C)	(A)	**0.4745**	0.4548	0.9103	0.0099
(R,C,M)	(R,M)	**0.4576**	0.4062	0.9359	0.0068
(C,M)	(M)	**0.4720**	0.4574	0.3325	0.0819
(R,C)	(R)	**0.4003**	0.3669	0.1925	0.1090

Table E.54: Wilcoxon test results for context inclusion and exclusion configurations of the DIT task for ZooKeeper v3.5.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,R,M)	**0.3776**	0.3706	$p < 0.01$	0.2224
(A,C,M)	(A,M)	**0.3667**	0.3643	0.9952	0.0005
(A,R,C)	(A,R)	**0.3985**	0.3858	$p < 0.01$	0.2520
(A,C)	(A)	0.3579	**0.3637**	0.9096	0.0080
(R,C,M)	(R,M)	**0.3189**	0.2970	$p < 0.01$	0.2206
(C,M)	(M)	**0.4247**	0.3188	$p < 0.01$	0.3944
(R,C)	(R)	**0.3456**	0.2973	$p < 0.01$	0.5620

Table E.55: Wilcoxon test results for message inclusion and exclusion configurations of the FLT task for ZooKeeper v3.5.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,R,C)	0.4788	**0.4818**	0.8554	0.0161
(A,C,M)	(A,C)	**0.4950**	0.4745	0.2021	0.1145
(A,R,M)	(A,R)	**0.4790**	0.4503	0.3914	0.0733
(A,M)	(A)	0.4531	**0.4548**	0.3149	0.0870
(R,C,M)	(R,C)	**0.4576**	0.4003	0.0599	0.1565
(C,M)	(C)	**0.4720**	0.3930	0.0474	0.1684
(R,M)	(R)	**0.4062**	0.3669	0.0243	0.1870

Table E.56: Wilcoxon test results for message inclusion and exclusion configurations of the DIT task for ZooKeeper v3.5.0 (*RQ 3.2*)

Configurations		MRRs		p-value	Effect size
(A,R,C,M)	(A,R,C)	0.3776	**0.3985**	$p < 0.01$	0.1945
(A,C,M)	(A,C)	**0.3667**	0.3579	0.6874	0.0281
(A,R,M)	(A,R)	0.3706	**0.3858**	0.0533	0.1342
(A,M)	(A)	**0.3643**	0.3637	0.4763	0.0499
(R,C,M)	(R,C)	0.3189	**0.3456**	$p < 0.01$	0.3760
(C,M)	(C)	0.4247	**0.4565**	$p < 0.01$	0.1747
(R,M)	(R)	0.2970	**0.2973**	0.2128	0.0859