Comparison of Data Preprocessing Techniques on Software Sources for Topic Modeling

Author	John Willems
Student Number	835669790
Date	April 22nd, 2014

Comparison of Data Preprocessing Techniques on Software Sources for Topic Modeling

Master Thesis

Master Software Engineering

Faculty of Management, Science & Technology

Open Universiteit (OUNL)

Author	John Willems
Student Number	835669790
Date	April 22nd, 2014
Chairman	prof. dr. M.C.J.D. van Eekelen
Supervisor	dr. B. Heeren
Course Number	T75317

Abstract

Studies have shown that topic modeling with Latent Dirichlet Allocation (LDA) is a useful (semi-)unsupervised technique to find topics in software source code that share latent commonalities and reveal information about the software system that was not known before. As topic modeling uses unstructured data we found no consensus in literature how to conduct data preprocessing on software source code to extract unstructured data. We define unstructured data as documentation, comments, string literals and programmer-defined names. In this thesis we want to find the data preprocessing technique that leads to the most optimal topic distribution for a given software system, therefore we create an experiment in which we compare four data preprocessing techniques. We select two techniques from literature, we define one by ourselves and we try one technique in which we take the software source code as-is. To measure the differences between the four techniques we use structural coupling metrics. We develop software that is dedicated to our experiment. We use the domain-specific language Rascal to develop software for data preprocessing and calculations. With the programming language Java we develop software for LDA topic generation with Gibbs sampling and word stemming. Results suggest there is minor difference between the four techniques when we perform the experiment for two software systems. This implies we can use the software source code as-is for topic modeling. If future work confirms this preliminary result it means a significant reduction of effort using topic modeling for software systems.

Keywords: Latent Dirichlet Allocation, Topic Modeling, Rascal, Metrics, source code

Table of Contents

Abstract v

Summary xi

Samenvatting xii

1 Introduction 1

1.1 Thesis Statement 1

1.2 Scope 2

1.3 Thesis Overview and organization 4

2 Domain-Specific Language Rascal 5

2.1 Introduction 5

2.2 Rascal's M3 model 7

2.3 Rascal Type System 9

2.4 Pattern matching 9

2.5 Rascal-to-Java Bridge 10

2.6 Comma-Separated Values 10

3 Probabilistic Topic modeling with LDA 11

3.1 Definitions of IR models 11

3.2 LDA algorithm with Gibbs sampling 12

4 Coupling Software Metrics 17

5 Data Preprocessing Tool 18

5.1 Definition Data Preprocessing Techniques 18

5.2 Data Preprocessing Techniques Implementation 19

6 Topic Generation 22

7 Coupling Metrics Calculations 25

8 Data Preprocessing Techniques Comparison and Result 27

8.1 Definition Inner and Outer Topic Coupling 27

8.2 Result 29

9 Threats to Validity 31

9.1 Verify result Metrics calculations with program CKJM 31

9.2 Rascal issues 34

9.3 Choice Coupling Metrics 34

10 Related Work 35

11 Conclusion and Future Work 37

11.1 Conclusion 37

11.2 Future work 38

12 Appendices I

13 Bibliography.....XIII

List of Tables

Table 1-1: Characteristics of our two systems, JHotDraw and jEdit.....3
Table 2-1: Rascal model M3 core relations.....8
Table 2-2: Rascal Basic Types.....9
Table 3-1: topic assignments of terms in document x.....13
Table 3-2: frequency of all terms from all M documents grouped by topic: posterior distribution.....13
Table 3-3: example Gibbs sampling subtracted 1 from entry "land" ,"crocodile".....14
Table 3-4: example Gibbs sampling replaced topic with question mark.....14
Table 3-5: example Gibbs sampling multiplication frequency topics and frequency term "crocodile".....14
Table 3-6: example Gibbs sampling updated topic distribution.....15
Table 3-7: example Gibbs sampling updated posterior distribution.....15
Table 5-1: Overview Definition Unstructured Text of the four Techniques.....19
Table 6-1: Stop-word list for each technique.....23
Table 9-1: Box Plot Figures Metrics Comparison jEdit.....32
Table 9-2: Box Plot Figures Metrics Comparison JHotDraw.....33

Illustration Index

Figure 1-1: Overview processes and data-flow.....4
Figure 2-1: EASY Paradigm.....6
Figure 2-2: Extract -Analyze-View paradigm.....7
Figure 3-1: Graphical model representation of LDA [2].....12
Figure 3-2: Process flow LDA topic generation.....15
Figure 6-1: Snapshot document-topic matrix θ of jEdit of the Raw technique.....24
Figure 8-1: abstract representation of topic assignment and document coupling metric relations.....28
Figure 9-1: jEdit comparison metrics CBO Rascal and CKJM tool.....32
Figure 9-2: jEdit comparison metrics CA Rascal and CKJM tool.....32
Figure 9-3: JHotDraw comparison metrics CBO Rascal and CKJM tool.....33
Figure 9-4: JHotDraw comparison metrics CA Rascal and CKJM tool.....33
Figure 12-1: Top-Words Selective technique. Left jEdit, right JHotDraw.....I
Figure 12-2: Top-Words Grant technique. Left jEdit, right JHotDraw.....II
Figure 12-3: Top-Words Thomas technique. Left jEdit, right JHotDraw.....III
Figure 12-4: Top-Words Raw technique. Left jEdit, right JHotDraw.....IV
Figure 12-5: Box Plot Ti metric with CBO for JHotDraw.....V
Figure 12-6: Box Plot Ti metric with CBO* for JHotDraw.....V
Figure 12-7: Box Plot Ti metric with DAC for JHotDraw.....V
Figure 12-8: Box Plot Ti metric with ATFD for JHotDraw.....VI
Figure 12-9: Box Plot Ti metric with CA for JHotDraw.....VI
Figure 12-10: Box Plot To metric with CBO for JHotDraw.....VII
Figure 12-11: Box Plot To metric with CBO* for JHotDraw.....VII
Figure 12-12: Box Plot To metric with DAC for JHotDraw.....VII
Figure 12-13: Box Plot To metric with ATFD for JHotDraw.....VIII

Figure 12-14: Box Plot To metric with CA for JHotDraw.....VIII
Figure 12-15: Box Plot Ti metric with CBO for jEdit.....IX
Figure 12-16: Box Plot Ti metric with CBO* for jEdit.....IX
Figure 12-17: Box Plot Ti metric with DAC for jEdit.....IX
Figure 12-18: Box Plot Ti metric with ATFD for jEdit.....X
Figure 12-19: Box Plot Ti metric with CA for jEdit.....X
Figure 12-20: Box Plot To metric with CBO for jEdit.....XI
Figure 12-21: Box Plot To metric with CBO* for jEdit.....XI
Figure 12-22: Box Plot To metric with DAC for jEdit.....XI
Figure 12-23: Box Plot To metric with ATFD for jEdit.....XII
Figure 12-24:Box Plot To metric with CA for jEdit.....XII

Appendices

Appendix A: Top-words per technique.....I
Appendix B: Box plots JHotDraw.....V
Appendix C: Box plots jEdit.....IX

List of Abbreviations and Symbols

ATFD	Access To Foreign Data
CA	Afferent Coupling
CBO	Coupling Between Objects
CK	Chidamber & Kemerer
CKJM	Chidamber and Kemerer Java Metrics
CSV	Comma-separated Values
CVE	Common Vulnerability and Exposure
CWI	Centrum Wiskunde en Informatica
DAC	Data Abstraction Coupling
DSL	Domain-Specific Language
EASY	Extract Analyze Synthesize Paradigm
IR	Information Retrieval
JDK	Java Development Kit
JDT	Eclipse Java Development Tool
JWNL	Java WordNet Library
LDA	Latent Dirichlet Allocation
ML	Machine Learning
MOOD	Metrics Object Oriented Design
MSR	Mining Software Repositories
PHP	Hypertext Preprocessor
pLSI	Probabilistic Latent Semantic Analysis
REPL	Read-Eval-Print-Loop
RTC	Relational Topic Coupling
RTM	Relational Topic Model
SDK	Software Development Kit
SQL	Structured Query Language
URI	Uniform Resource Identifiers
XML	Extensible Markup Language
α	Hyper parameter document-topic distribution
β	Hyper parameter topic-term distribution
θ	Document-topic matrix
φ	Topic-term matrix

Summary

Information Retrieval (IR) techniques are developed to handle unstructured data which makes them a useful technique to discover relationships in software repositories between different artifacts and that can help to understand a software system. In particular, topic modeling was found to be very useful because it enables users/researchers to discover latent relations between documents in a fully automated (semi-)unsupervised way. One of the most popular techniques for probabilistic topic modeling is Latent Dirichlet Allocation (LDA), which assigns topics to documents with a certain probability.

We define unstructured data in software source code as documentation, comments, string literals and programmer-defined names. In our research about studies which apply LDA on software source code we found no consensus in data preprocessing techniques to extract the unstructured data from software source code.

In this research we want to find a data preprocessing technique, that leads to the best topic distribution for a given software system, therefore we create an experiment in which we compare four data preprocessing techniques. We select two techniques from literature, we define one by ourselves and we try one technique in which we take the software source code as-is.

To measure the differences between the data preprocessing techniques, we use the structural coupling metrics CBO, CBO*, DAC, ATFD and CA. Each metric measures a different kind of coupling. We develop a data preprocessing and metrics calculation tool in the domain-specific language (DSL) Rascal and is dedicated to our experiment. In Java we implement word stemming with Wordnet and for LDA topic generation with Gibbs sampling we use the Java framework Mallet. We perform the experiment for the software systems JHotDraw and jEdit.

Results suggest that there is a minor difference between the four data preprocessing techniques. This implies we can use the software source code as-is for topic modeling. If future work confirms this preliminary result it means a significant reduction of effort using topic modeling for software systems.

Samenvatting

Information Retrieval (IR) technieken zijn ontwikkeld om met ongestructureerde gegevens om te gaan, waardoor ze een geschikte techniek blijken te zijn om relaties in software repositories te ontdekken. Deze technieken kunnen een bijdrage leveren aan een beter inzicht in de samenhang van de artefacten in een software repository. Een van deze IR technieken is Topic Modellering die gebruikers en onderzoekers in staat stelt om latente relaties in software systemen te ontdekken. Eén van de meest populaire topic modeleringsalgorithmen is Latent Dirichlet Allocation (LDA), wat een probabilistisch model is en topics aan documenten toewijst met een bepaalde waarschijnlijkheid.

In onze studie, waarbij LDA wordt toegepast op de broncode van een software systeem, hebben we geen consensus kunnen vinden hoe ongestructureerde data te extraheren van software broncode. Hierbij definiëren we ongestructureerde data als documentatie, commentaar, string literals en naamgeving zoals die door de ontwikkelaar is bepaald.

In dit onderzoek willen we een data extractietechniek vinden die de extractie van de onstructureerde data zodanig selectief uitvoert, dat hierdoor de beste verdeling van topics naar documenten plaatsvindt. We vergelijken vier data extractietechnieken met elkaar. Twee technieken selecteren we uit de literatuur, we creëren een techniek zelf en als vierde techniek nemen we de broncode in zijn geheel als ongestructureerde data. Om de verschillen tussen de vier technieken te kunnen meten, gebruiken we de structurele koppeling metrieken CBO, CBO*, DAC, ATFD en CA die ieder een ander aspect van koppeling meten.

Specifiek voor ons experiment ontwikkelen we software. In de domein-specifieke taal Rascal ontwikkelen we software die de ongestructureerde data uit de broncode extraheert en de metriek koppeling berekeningen uitvoert. Voor LDA topic generatie met Gibbs sampling ontwikkelen we software in Java met gebruikmaking van het Java framework Mallet. Eveneens in Java ontwikkelen we functionaliteit die werkwoorden tot hun stam terugbrengt en zelfstandige naamwoorden tot hun enkelvoudsvorm. De software systemen waar we ons experiment op uitvoeren zijn JHotDraw en jEdit.

De voorlopige resultaten suggereren dat er een minimaal verschil is tussen de vier data extractietechnieken. Daarom is het extraheren van ongestructureerde data uit broncode overbodig en kan topic modellering meteen op broncode worden toegepast. Verder onderzoek, waarbij meerdere software systemen worden betrokken, moet dit resultaat bevestigen.

1 Introduction

We want to organize a corpus, let's say of 10.000 documents, in 100 topics. Reading all the documents and assigning them to the appropriate topic is a very labor and time intensive task. A more convenient and less time-consuming way is to use an algorithm, assuming the documents are available in a digital format. In the field of Information Retrieval (IR) topic modeling is a technique to reveal latent information from unstructured data in an automated way and assign documents to topics [1,2,17]. Within topic modeling several techniques are available, such as probabilistic Latent Semantic Indexing (pLSI) [1] and Latent Dirichlet Allocation (LDA) [1,2]. We will use the latter in this research because pLSI provides no probabilistic model at the document level [2].

Software systems are getting increasingly larger and more complex [8,10,17,19], therefore it is challenging for software developers and project managers to get an overview of the software system at hand. Research in software engineering has made significant progress in mining and analyzing software repositories in a structured manner [8,17,19]. An automated technique to get an understanding of latent relations between software source code (source code) files of a software system was missing until in 2007 Linstead et al. [19] published a paper on how to apply topic modeling on software systems using LDA, showing the effectiveness and usefulness of this technique. Other researchers followed Linstead et al. investigating the possibilities of topic modeling for software systems [7,8,15,17]. In general, IR models are now subject of research in mining software repositories, like concept mining, constructing source code search engines, recovering traceability links between artifacts (for example between developer's emails and source code, or between a bug database and source code), calculating source code metrics and clustering similar documents [17].

1.1 Thesis Statement

The unstructured data in software source code is stored as comments, documentation, string literals and programmer-defined names [7,17]. To our knowledge and of Thomas [17, page 49], no study is conducted to determine a data preprocessing technique to extract the unstructured data from source code which leads to a topic distribution that is useful for practitioners. The research we studied of Thomas [17], Lindstead et al.

[19] and Grant et al. [7] we found no consensus in data preprocessing. For example, Thomas separates the programmer-defined names and filters out the Java program language[1] keywords, applies word stemming and the common English stop-word list, Grant et al. just take the methods and separate the programmer-defined names but do not mention anything about the program language specific keywords and stop-words, Lindstead et al. just remove the common English stop-words and the names of all classes of the Java SDK. The latter is different from the Java language keywords, as Thomas mentions. We also consider the Raw technique in which we take the Java source code files as-is, without any data preprocessing, as an extreme opposed to the other three techniques.

Our hypothesis:
Based on intuition we think we can do better as Thomas and Grant and describe the data preprocessing technique Selective: we separate the unstructured data (documentation, comments, string literals and programmer-defined names) through data preprocessing from source code, separate the programmer-defined names in components, use a common English stop-words list and apply stemming to the terms. To determine if our technique results to a better topic distribution, compared to the technique described by Thomas and Grant and the Raw technique, we measure the results of the four techniques with structured coupling metrics.

1.2 Scope

We develop a tool for our experiment in the Domain-Specific Language (DSL) Rascal[2] which enables us to do data preprocessing, calculate coupling metrics and to compare the results between the four data preprocessing techniques. We use the programming language Java to develop tools for word stemming and for topic generation. For the former we use WordNet[3] (a lexical database for English) and Java software library JWNL[4], and for the latter we use the Java software framework Mallet[5]. The tool will simulate the data preprocessing technique of Thomas [17] and Grant et al. [7], because

1 http://www.oracle.com/us/technologies/java/overview/index.html
2 *http://www.rascal-mpl.org/*
3 *http://wordnet.princeton.edu/*
4 *http://sourceforge.net/apps/mediawiki/jwordnet/index.php*
5 *http://mallet.cs.umass.edu/*

their tools are not available in a usable format for us. We perform our experiment on the two software systems JHotDraw[6] and jEdit[7].
In table 1-1 we list the main characteristics of the two software systems.

Characteristics	***JHotDraw***	***jEdit***
Purpose	Drawing Framework	Text Editor
Implementation Language	Java	Java
License	Open Source	Open Source
Release	6.0 beta 1	5.1.0
Number of Source Code Files	309	535
Lines of Code (thousands)	57	172
Number of terms (thousands)	619	830

Table 1-1: Characteristics of our two systems, JHotDraw and jEdit.

In figure 1-1 we depict an overview of the different processes of our experiment and storages for intermediate data, with chapter and section reference where we outline the process. We store intermediate data because it simplifies the development and it enables us to execute each process in isolation. The processes execute in a defined sequence as the arrows indicate.

[6] *http://sourceforge.net/projects/jhotdraw/*
[7] *http://sourceforge.net/projects/jedit/*

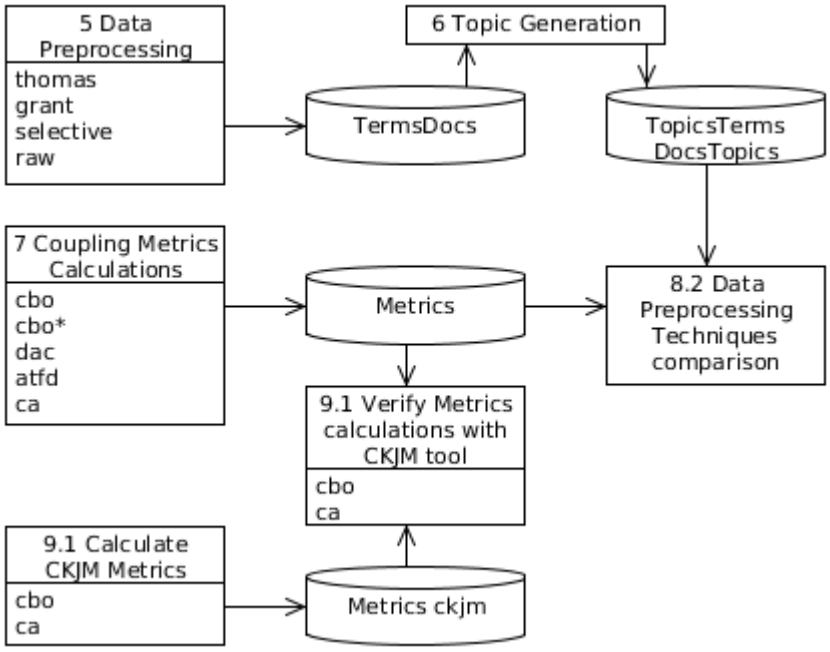

Figure 1-1: Overview processes and data-flow

The tool is solely built for the purpose of the experiment, therefore we put no effort in making the tool scalable or usable for other purposes. On the other hand we put much effort in the reliability and validity because the results highly depend on certain conditions and parameters.

1.3 Thesis Overview and organization

In chapters 2 until 4 we provide a theoretical background of the techniques and methods we use in the research outlined in this thesis. Chapter 2 covers a general introduction of the Domain-Specific Language Rascal with details of language features we use in the tool that we develop for our experiment. In chapter 3 we provide a theoretical overview of LDA with Gibbs sampling and a comprehensive example. In chapter 3 we give a definition of the coupling metrics we use in our experiment.

In chapters 5 until 9 we outline the experiment which we conduct. Chapter 5 describes the tool we develop for data preprocessing. In chapter 6 we describe the Topic Generation implementation. Chapter 7 the Coupling metrics Calculations. In chapter 8 we define the data preprocessing techniques comparison and the results.

Chapter 9 covers the threats of validity and chapter 10 related work. In final chapter 11 our conclusion and future work.

2 Domain-Specific Language Rascal

In this chapter we give in section 2.1 an introduction of the Domain-specific Language Rascal. In section 2.2 we outline the Rascal M3 model. In section 2.3 the Rascal type system. Section 2.4 covers Rascal's pattern matching and section 2.5 the Rascal-to-Java bridge to call Java programs. The final section 2.6 covers how Rascal handles Comma-separated Values files.

2.1 Introduction

Metaprogramming is developing software programs that manipulate source code or generate new source code [16], in which the source code is the executable specification of a software system and thus exhibits behavior [17]. Source code can be treated as data, therefore every program language with file access features could be used as a metalanguage. In such a case the steps performed are:

a) read the source code
b) analyze the source code
c) modify and/or create new code
d) save the source code
e) compile to an executable format
f) execute the program and observe the new behavior.

Metaprogramming in multipurpose program languages is troublesome due to the lack of features specific for this purpose, therefore a metaprogramming language with focus on the implementation of a Domain-Specific Language (DSL) for rapid construction of software analysis and software transformation is advisable. Rascal is such a DSL [16] and therefore we use Rascal for our research.

Rascal implements the EASY paradigm, depicted in figure 2-1, in which we see the same steps as listed above from a till f. The data store “?” indicates the system-of-interest and can be anything. For example, software source code or files with comma-separated values. In particular, access to the former is supported by the Rascal Eclipse[8] plug-in, which enables access to the Eclipse Java Development Tool (JDT)[9]

[8] http://www.eclipse.org/
[9] http://www.eclipse.org/jdt/

from the Rascal development environment. The Eclipse plug-in also supports Read-Eval-Print-Loop (REPL) for testing purposes.

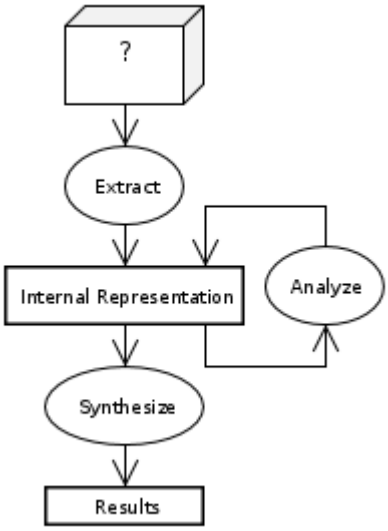

Figure 2-1: EASY Paradigm

The language Rascal is easy accessible for developers who are familiar with the program languages C++ and/or Java, or in general with an imperative programming language with a static type system [16]. Particularly to the same concepts in other programming languages the primary data types, structured control flow and the exception handling are similar. To make Rascal even more easy accessible it has a layered design. As a developer gets more comfortable with the language she is free to use more advanced features.

In this research we are only interested in the analysis of the source code and not in modifying or generating new source code, therefore we use the Extract-Analyze-View paradigm [16] as depicted in figure 2-2. This is a sub-set of the EASY paradigm because the synthesis is omitted.

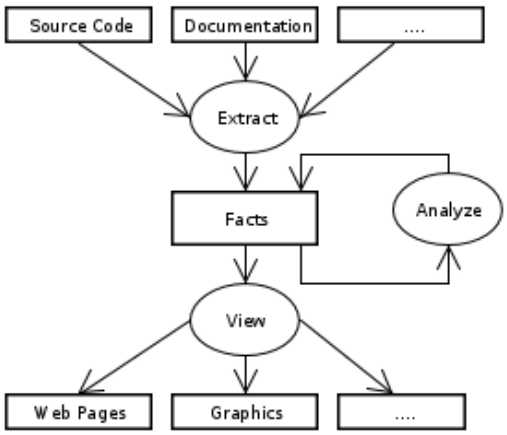

Figure 2-2: Extract -Analyze-View paradigm

2.2 Rascal's M3 model

The process "Extract", in figure 2-2, will store a representation of the source code in the internal Rascal structure M3 [11], which is a unified model for storing facts about programs. Software projects have to be available as an Eclipse project in compilable source code, as a prerequisite for Rascal M3 model. The M3 structure keeps data in an immutable, typed form, which can be directly produced, manipulated and analyzed with the Rascal primitives. We describe two elements of the M3 structure which are important for this research. These elements are locations and relations.

For locations the URI is defined as

```
|<scheme >://<auth>/<path>?<qry>|(off>,<len>)
```

The URI keeps an information link back to the source code. The location can be physical or logical. We use only the latter in which the logical location is the Java Eclipse project. A typical example for this scheme is:

```
loc project = |project://JEDIT|;
```

When the project is defined the M3 data structure can be derived from the project, for example:

```
M3 m3Model = createM3FromEclipseProject(project);
```

M3 defines a naming scheme for every source code element within the project. For example, the former is a Java class and the latter a method element.

```
|java+class:///org/ ... /textarea/ChunkCache|
|java+method:///org/... /Macros/getMacro(java.lang.String)|
```

Other examples of occurrences for scheme are “java+primitives”, “java+constructor” and “java+compilation Unit”. The latter is the equivalent of a Java file. Since we use only Java projects in the research the schemes start with “java”.

The M3 model contains core relations between code locations in pairs, for instance:

```
<|java+class:///org/View|,|java+field:///org/View/prev|>
```

denotes a binary relation between a class and a field. The full list of relations covered by Rascal is given in table 2-1.

All M3 relations can be produced, manipulated and analyzed using Rascal primitives.

M3 relation <x,y>	Description
fieldAccess	Attribute x access field attribute y
extends	Attribute x extends attribute y
typeDependency	Attribute x depends on the type of attribute y
methodInvocation	Attribute x makes an invocation on method attribute y
containment	Attribute x contains attribute y
messages	Attribute x contains string literals y
names	Attribute name is located at attribute y
documentation	Attribute x has documentation within attribute y
implements	Attribute x implements interface attribute y
uses	Attribute x (project level) makes use of attribute y
annotations	Attribute x has annotation attribute y
methodOverrides	Attribute x is overrides attribute y
modifiers	Attribute x has modifier y
types	Attribute x has parametrized type y
declarations	Attribute x is declared by attribute y

Table 2-1: Rascal model M3 core relations

2.3 Rascal Type System

The Rascal type system [12] is based on a type lattice with void at the bottom and value at the top. The latter is super-type of all types. In between are the atomic values (bool, int, real, str, loc and datetime), type of tree values and composite types. Example of composite types are list [&T], set [&T], tuple [&T,&T], rel[&T,&T] and lrel[&T,&T]. Sub-typing is always covariant; return types of methods are also covariant and argument types are contravariant [12]. Custom types can be defined through type literals. For example

```
alias customType = lrel[str name1, str name2]
```

In table 2-2 we depict the Rascal basic types [16].

Type	Example
bool	true, false
int	1, 0, -1, 3245422
real	1.0, 1.043e10, -4.23
str	“abc”
loc	|java+class:///org/gjt/sp/jedit/View|
tuple[t_1, .. t_n]	<1,2>,<”abc”,5,8>
list[t]	[], [1,2,3],[“abc”,2,5.0]
set[t]	{}, {“abc”,3,4},
rel[t_1, ... t_n]	{<1,2,3>,<4,5,6>}
lrel[t_1, ... , t_n]	[<1,2,3>,<4,5,6>]
map[t,u]	(), (1:true, 2:false), (6:{1,2,3})
node	F, add(x,y), g(“abc”, [1,2,3])

Table 2-2: Rascal Basic Types

2.4 Pattern matching

Pattern matching is provided against all data types [12,16]. Furthermore, Rascal supports deep matching (/), negative matching (!), set matching and list matching. The patterns can be used in switch-case statements and comprehensions. Rascal is not using groups as common with regular expressions, but the notation <name:expression> in which the result of the expression is assigned to name. In the case there are multiple matches the backtracking works from right to left.

2.5 Rascal-to-Java Bridge

In order to make use of programs developed in Java, Rascal is enriched with a Rascal-to-Java bridge [12]. For example, the following fragment shows a typical example to call a Java method.

```
@javaClass{nl.ou.stemming.Stemmer}
java str stemWord(str arg);
```

The instantiation of the class is achieved with the @javaClass annotation. The second line invokes the method. The return type is a Rascal type, in the example type str. The return type, in the invoked Java method, is of type IvalueFactory and constructs the required Rascal value. If the return types do not match an exception is thrown.

2.6 Comma-Separated Values

Rascal supports reading and writing of CSV data from and to files [12]. In writing and reading of files, type literals are of great use.

For example:

```
alias aliasResult = rel[str name1, str name2];
aliasResult nameResult = { ... };
writeCSV(nameResult,|file:///filedir/fileName.csv|);
```

Since the variable nameResult is of type aliasResult, the CSV file contains as first row the column names name1 and name2. The other way around works in the same way, for example:

```
aliasResult fileContent =
      readCSV(#aliasResult,|file:///filedir/fileName.csv|);
```

The type of value fileContent is of aliasResult. In this way Rascal provides type safety with external sources.

3 Probabilistic Topic modeling with LDA

Probabilistic topic models, one of the basic models of IR, have their origin in Machine Learning (ML) and include a set of algorithms that aim at detecting and annotating large archives of thematic information. Topic models are based on the idea that documents are a mix of topics, where a topic is a probability distribution over the words [1,17].

Before going into the details of probabilistic Topic Modeling with LDA we give in section 3.1 the definitions of the common vernacular in IR. In section 3.2 we describe the LDA algorithm with Gibbs sampling with a comprehensive example.

3.1 Definitions of IR models

This section describes the terms and common definitions within the field of IR models [2,17]. We will use these in the remainder of this thesis.

A **term** w is string of one or more alphanumeric characters. Terms are not unique in a document.

A **document** is an ordered sequence of N terms denoted by $w = (w_1 \ldots w_N)$ where w_n is the n^{th} term in the sequence.

A **corpus** is an unordered set of M documents denoted by $d = \{w_1 \ldots w_M\}$

A **vocabulary** is an unordered set of m terms in C denoted by $V = \{v_1 \ldots v_m\}$

The **term-document matrix A** is an $m \times M$ matrix whose i,j entry is the number of occurrences of term w_i in document d_j.

A **topic z** is an m-length vector of probabilities over the vocabulary V. The total numbers of topics is K.

The **document-topic matrix θ** is a $M \times K$ matrix whose i,j entry is the probability of topic z_j in document d_i.

The **topic-term matrix φ** is a $K \times m$ matrix whose i,j entry is the probability of term w_j in topic z_i.

3.2 LDA algorithm with Gibbs sampling

LDA is a probabilistic topic modeling method and assigns documents to multiple topics, each with a certain probability [2].

The LDA algorithm is depicted in a graphical model in figure 3-1. The M area represents the documents and the N area the terms and each plate can be interpreted as a for-each loop. The hyper-parameter α is used for the document-topic distribution and hyper-parameter β for the topic-term distribution. Both α and β are always > 0. The other variables are defined in the previous section.

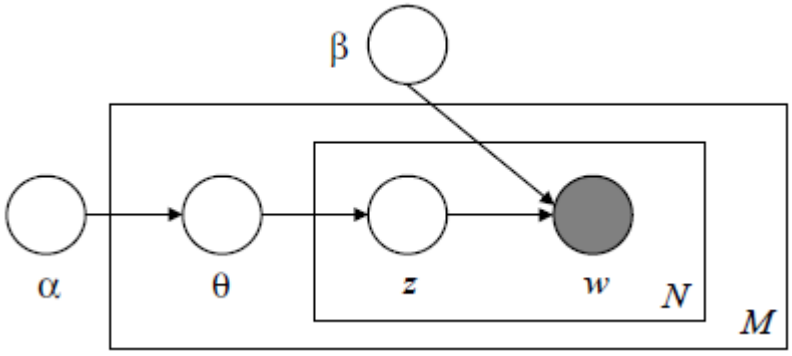

Figure 3-1: Graphical model representation of LDA [2]

We explain the working of the LDA algorithm with an example. If we want to organize 10.000 documents manually we create a list of 100 topics in advance. The latter is always the case and the number of topics is defined by K. In our example the list could start with the topic labels “water”, “land”, “air” etc. Next we have 100 highlighters in different colors. Each color represents a topic. When reading a document x we highlight the key terms with the corresponding topic color. For example the term “tiger” is assigned to the topic “land”, but there are also terms assigned to multiple topics. For example, the term “crocodile” belongs to the topic “water” but also to the topic “land”. Therefore, the term “crocodile” is highlighted by two different highlighters what is also the case for term “swan”. We do this for all the terms in document x and end up with a topic-term distribution depicted in table 3-1.

Topic z	land	water	land	land	air	water
Term w	swan	crocodile	crocodile	tiger	swan	swan

Table 3-1: topic assignments of terms in document x

We do this for all the documents in the corpus and omit the terms such as “the”, “a”, “or” etc. because they contain no semantic information and cannot be allocated. When done with reading, we group all the terms by color and remember the source document of each term. An example is depicted in table 3-2 which is called the posterior distribution.

	water	***land***	***air***
tiger	0	6	0
crocodile	4	4	0
swan	1	2	6

Table 3-2: frequency of all terms from all M documents grouped by topic: posterior distribution

By now, we know which topic belongs to which documents and we know the terms of each topic.

Topic modeling algorithms fall into two categories. Sampling-based algorithms and variational algorithms [2]. We will ignore the latter and mention it only for completeness. Sampling-based algorithms collect samples from the posterior distribution to approximate it with an empirical distribution. We use Gibbs sampling as the sampling-based algorithm [1,7,17] and explain Gibbs sampling with the use of the posterior distribution as depicted in table 3-2 by applying the following computation:

Initialization topic assignments randomly
for each iteration (issued by parameter, for example 10.000)
    for each document
        for each word
            re-sample topic for word, given all other words and their
            current topic assignment
produce repost

We give an example, derived from a presentation given by David Mimno[10], for one iteration and refer to the data in tables 3-1 and 3-2. We reassign the term “crocodile” of topic “land”.

We subtract 1 from the entry of column “land” and row “crocodile” as depicted in table 3-3.

	water	***land***	***air***
tiger	0	6	0
crocodile	4	**3**	0
swan	1	2	6

Table 3-3: example Gibbs sampling subtracted 1 from entry "land" ,"crocodile"

For the topic distribution of document x we change topic “land” in a question mark for term “crocodile” as depicted in table 3-4.

z	water	water	**?**	land	air	land
w	swan	crocodile	crocodile	tiger	swan	swan

Table 3-4: example Gibbs sampling replaced topic with question mark

Subsequently we look for the term crocodile in another topic. First we look for the frequency of the topics in document x. Topic “land” occurs twice, “water” once and “air” once. Secondly we look how many times the term “crocodile” occurs in the posterior distribution. Then we multiply the values with the hyper-parameters. As we can see, in table 3-5, the number of occurrences of crocodile for topic “air” is zero. In this case the hyper-parameter β is used, because in the multiplication no terms are allowed to be zero. In case the number of topics for this document equals zero the hyper-parameter α is used (for convenience we set the hyper-parameters α and β both to 1 in this example).

	Water	***Land***	***air***
crocodile	(2 * α) * (4 * β) = 8	(1 * α) * (4 * β) = 4	(1 * α) * β = 1

Table 3-5: example Gibbs sampling multiplication frequency topics and frequency term "crocodile"

[10] http://journalofdigitalhumanities.org/2-1/the-details-by-david-mimno/

Next we search for the topic which has the highest probability and assign the term crocodile to this topic. From the results we conclude this is topic “water” and update the topic distribution for document x. This is depicted in table 3-6.

z	water	water	**water**	land	air
w	swan	crocodile	crocodile	tiger	swan

Table 3-6: example Gibbs sampling updated topic distribution

Also we update the posterior distribution for the final result in the iteration of our example. To the number of occurrences for “crocodile” for the topic “water” 1 is added. The new value is 5. This is depicted in table 3-7.

	Water	***Land***	***air***
tiger	0	6	0
crocodile	**5**	3	0
swan	0	2	6

Table 3-7: example Gibbs sampling updated posterior distribution

As depicted in figure 3-2 the process flow from corpus to the matrices θ and φ the parameters K, α and β are together with the term-document matrix A the input of the LDA algorithm, therefore LDA is a semi-unsupervised algorithm. The three parameters have significant influence on the populations of matrices θ and φ.
Research shows that the K-factor is difficult to determine [2,7,15,17]. If the k-factor is too low, terms are assigned to topics they not really belong to. On the other hand if the K-factor is too high terms are shattered over topics and semantic information is lost.

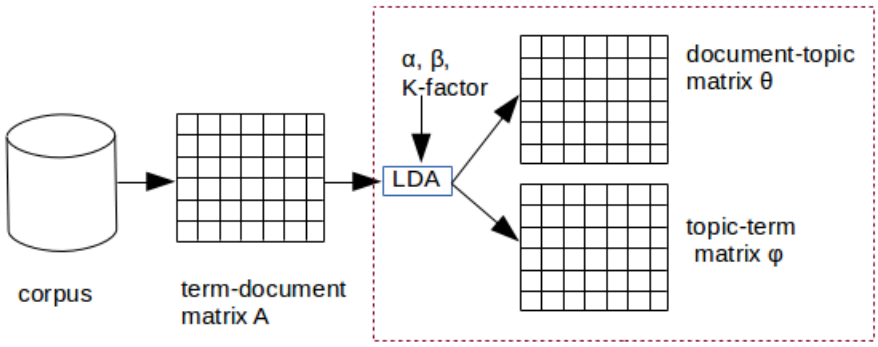

Figure 3-2: Process flow LDA topic generation

Research has been conducted to find optimal values for the hyper-parameters α and β [15], but future work is needed. A high value of α results in a better smoothing of the topics for each document. A high value for β results in a more uniform distribution of terms per topic [15]. The most optimal distribution is when documents are assigned with a probability of nearly 100% to one topic and topics consist of terms with a probability of nearly 100% and other terms from the vocabulary with a probability of nearly 0%. In practice this is nearly impossible [1,17].

4 Coupling Software Metrics

Extensive research has been conducted to quantify the quality of software [5,9,13], which has resulted in metric suites such as the CK suite [5] and the MOOD suite [9]. Software metrics are very useful because they can provide in an automated way quality information about a software system [5,9,13]. A good deal of metrics can be found in the suites, but we are only interested in software coupling metrics because they enable us to compare the different techniques of extracting unstructured data from source code as input for topic modeling.

De Souza and Maia [13] provide five coupling metrics to measure the coupling between classes in a software system and show there is a correlation regarding coupling between classes of software systems which belong to the same category. Examples of categories used in the study are games, development and audio & video. De Souza et al. have selected these metrics because each of the metrics shows another aspect of the coupling between classes. The metrics are:

a) CBO (Coupling Between Objects) from the CK suite. This metric measures if there exists a reference from this class to another class. A reference is defined by the use of an attribute or a method invocation. A reference between two classes is only counted once.
b) CBO* is a variant of CBO: this metric counts the number of references from this class to the attributes of the other class and the number of method invocations of the other class.
c) DAC (Data Abstraction Coupling): counts the number of attributes declared with the type of another class.
d) ATFD (Access To Foreign Data): this metric counts the number of accesses of attributes from this class to other classes, directly or via accessor methods.
e) CA (Afferent Coupling): this metric counts the number of other classes that refer to this class. Actually, it is the inverse of CBO*.

5 Data Preprocessing Tool

In this chapter we present the data preprocessing tool we built for the four data preprocessing techniques. First we give in section 5.1 the definition of the four techniques. In section 5.2 we describe the implementation of the data preprocessing tool.

5.1 Definition Data Preprocessing Techniques

We define the four data preprocessing techniques as follows (the text in italic is copied unmodified from the corresponding research):

> ***Grant**: Given a source code corpus, a simple parse extracts the terms of interest from each document. In our case, these are programming language keywords, and programmer-defined names. Since many programmer-defined names have internal structure, we separate such names in their component pieces if they have been built in one of the standard ways (for example, breaking at underscores) and count the entire name and all of its sub-pieces as terms. For example, the term get_attribute would be represented by three terms: get, attribute and get_attibute. ….. Each source code package was segmented into individual methods or functions without comments using TXL.* [7]
>
> ***Thomas**: Before topic models are applied to source code, several preprocessing steps are generally taken in an effort to reduce noise and improve the resulting topics.*
>
> - *Characters related to the syntax of the programming language (e.g., "&&", "->") are removed; programming language keywords (e.g., "if", "while") are removed.*
> - *Identifier names are split into multiple parts based on common naming conventions (e.g., "oneTwo", "one_two").*
> - *Common English-language stopwords (e.g., "the", "it", "on") are removed.*
> - *Word stemming is applied to find the root of each word (e.g., "changing" becomes "chang").*
> - *In some cases, the vocabulary of the resulting corpus is pruned by removing words that occur in, for example, over 80% of the documents or under 2% of the documents.*

The main idea behind these steps is to capture the semantics of the developers' intentions, which are thought to be encoded within the identifier names and comments in the source code. The rest of the source code (i.e., special syntax, language keywords, and stopwords) are just noise and will not be beneficial to the results of topic models. [17][11]

Raw: we take the text from the Java files as-is and do no parsing, no stemming and no separation of programmer-defined names in components.

Selective: We selectively separate all unstructured data from the source code which consists of two parts: We take the comments, documentation and string literals, which we extract with the use of regular expressions. The second part consists out of programmer-defined names, including type names. We apply separation of programmer-defined names on all unstructured data (we assume programmer-defined names are also present in the comments, documentation and string literals). Finally we remove the common English-language stop-words from the vocabulary.

5.2 Data Preprocessing Techniques Implementation

For all four techniques we omit the copyrights information at the top of all files because the text is identical in every Java source file for the software system JHotDraw, therefore the text adds no additional semantic information for topic modeling. For the software system jEdit we have found several versions of the copyrights text but these texts are similar to each other. Before we describe the different techniques we give in table 5-1 an overview of the definition of unstructured data for each technique.

Technique	***Definition Unstructured Text***
Selective	• Comments, documentation and string literals • Decompose programmer-defined names in terms • Stemming of terms
Thomas	• Whole source code • Decompose programmer-defined names in terms • Stemming of terms
Grant	• Source code of methods • Decompose programmer-defined names in terms • programmer-defined word as term
Raw	• Whole source code

Table 5-1: Overview Definition Unstructured Text of the four Techniques

[11] We omit pruning because of time constraints of the project.

For the technique Selective we use Rascal pattern matching to extract the comments, documentation and string literals from the source code as opposed to the techniques Grant, Raw and Thomas where we simply read the Java files as a whole. We use the Rascal M3 language independent meta model (see section 2.2) to extract facts from the source code. The annotation relation m3@containment contains the relations between attributes. For example, the relations expressed in the pairs <Class A, Field F1>, <Unit 1, Class A> and <Class A, Method m> express that the Field F1 is part of Class A, Class A is part of Unit 1 and Method m is part of Class A. We choose the compilation unit as the root level and relate all attributes to this level in the following way: select all attributes with first element in the pair equal and lower than the compilation unit and take the transitive closure of the relations. From the result we select all relations with first pair element compilation unit, Referring to the example, the result is the pairs <Unit 1, Class A>,<Unit 1, Field 1> and <Unit 1, Method m>. This enables us to collect all attributes for each compilation unit.

We use Rascal pattern matching to decompose programmer-defined names in component pieces. For example, the programmer-defined camel-case class name “DefaultInputHandlerProvider” taken from jEdit is decomposed into the terms “Default”, “Input”, “Handler” and “Provider”. The same applies to programmer-defined names containing dashes or underscores.

The programmer-defined component pieces are merged with the comments, documentation and string literals and we apply word stemming on all terms. The software library we use is WordNet in combination with the library JWNL. Rascal invokes the Java program we developed. This is possible because Rascal provides a Rascal-to-Java bridge as outlined in section 2.5 Finally the terms are stored in a comma-separated values file.

For the technique Grant only the methods and constructors from the source code are taken in account and we use the annotation relation m3@containment to merge the attributes to the level of compilation unit and decompose the programmer-defined names in the following way. We take the same example as for technique Selective. Class name “DefaultInputHandlerProvider” is decomposed in the terms “Default”, “Input”, “Handler”, “Provider” and “DefaultInputHandlerProvider”. The terms are different from the technique Selective because we keep also the programmer-defined

name as term. We store the terms per compilation unit in a comma-separated values file.

For the technique Thomas we read the Java files, decompose the programmer-defined names as we do in the technique Selective, apply stemming to the terms and finally store the terms in a comma-separated values file.

For the technique Raw we read the Java files and store the terms, without any modification, in a comma-separated values file.

6 Topic Generation

Topic generation is implemented in Java with making use of the Java framework Mallet. The process takes the comma-separated values file produced in the previous process Data Preprocessing (see figure 1-1), as input and the following parameters (see Figure 3-2: Process flow LDA topic generation):

- number-of-topics K = 45 for both JHotDraw and jEdit.
- Number-of-iterations the algorithm will perform is set to 10.000
- number-of-top-words is set to 10.
- optimize-interval is set to 10
- burn-in-period is set to 300
- alpha and beta parameters are set to their defaults
- stop-word list is the Mallet Common English stop-word list, extended for technique Thomas with the Java keywords. For techniques Grant and Raw the stop-word list is omitted.

The comma-separated values file, the output from the previous process, is by Mallet internally transformed in a term-document matrix A (see section 3.1 for definition).
For number-of-topics we use the K-factor of 45 for both jEdit and JHotDraw as mentioned by Thomas [17, page 151]. Although software system jEdit is three times the size of JHotDraw (see table 1-1), we find realistic topics with a K-factor of 45 for jEdit. Figure 12-1 in Appendix A depicts the top-words of technique Selective for jEdit and JHotDraw. For jEdit, on the left side of figure 12-1, we can recognize topics specific for a text editor. For example, topic 0 deals with file handling and topic 6 deals with tables. On the right side, JHotDraw, topic 0 deals with operations on figures and topic 1 with the construction of figures. Figures 12-2 till 12-4 in appendix A depict the top-words of the other three techniques for JHotDraw and jEdit.

Grant et al. [7,8] propose to calculate the K-factor based on the number of documents with the formula: $t(x)=7.25*x^{0.365}$ where x is the number of documents. Based on this formula the K-factor for JHotDraw is 59 and for jEdit 72, which is higher than Thomas proposes. Noting that Grant et al. takes the methods in source

code as documents, this implies the number of documents is much higher than the number of Java source code file which we use as documents.
We take the value for number-of-iterations from Thomas [17, page 151], which is 10.000. The values of the optimize-interval, the burn-in-period and number-of-top-words are chosen based on the guidelines posted on the Mallet website. The optimize-burn-in-period is the number of iterations before the hyperparameter (section 3.2) optimization starts. The optimize-interval turns on the hyper-parameter optimization and holds the number of iterations optimization is applied. The parameter number-of-top-words holds the number of top words stored per topic in a file for topic inspection. See figure 12-1 until 12-4 in Appendix A for the top-words per data preprocessing technique for jEdit and JHotDraw.

Table 3-3 shows which stop-word list is used, if any, for the four techniques.

Technique	***Stop-words list***
Selective	Mallet Common English stop-word list[12]
Thomas	Mallet Common English stop-word list Java Language Keywords[13]
Grant	No stop-word list
Raw	No stop-word list

Table 6-1: Stop-word list for each technique

For the Selective technique Java language keywords are only in the term-document matrix A when these keywords are part of the unstructured data.
In case of technique Thomas the term-document matrix A includes all the source code, hence also the Java language keywords. The use of the stop-word list, as listed in table 6-1, will remove these Java language keywords from the vocabulary before generating the topics but also from the comments, documentation, string literals and decomposed programmer-defined names.
For the techniques Raw and Grant no stop-words list is used and all Java language keywords are part of the vocabulary. In figure 12-2 Appendix A we depict the top-words for both jEdit and JHotDraw for technique Grant. We see remarkably few

[12] https://github.com/jmcejuela/mallet/blob/master/src/cc/mallet/pipe/TokenSequenceRemoveStopwords.java
[13] http://docs.oracle.com/javase/tutorial/java/nutsandbolts/_keywords.html

Java language keywords and no programmer-defined names in the topics. In general the topics make sense.

Because of the size of the file we show in figure 6-1 only a small fraction of the document-topic matrix θ of technique Raw.

```
0 CircleFoldPainter.java    15 0.4496390672   4  0.4000455256  33   0.052809984  27   0.0519809974
1 AboutDialog.java          21 0.5183804607  44  0.1739738039   4 0.0836362029   2  0.0736105911  24  0.070673248
```

Figure 6-1: Snapshot document-topic matrix θ of jEdit of the Raw technique

The first column denotes the document number, and the second column denotes the document name where we omit the pathname for clarity. The third column and further denote the topic and the probability of the topic for this document. For document 0 with name CircleFoldPainter.java topic 15 has a probability of 44%, topic 4 a probability of 40% etc.

7 Coupling Metrics Calculations

In chapter 4 we defined the coupling metrics we use for this research. In this chapter we describe how to calculate these coupling metrics with Rascal. As the compilation unit is the root level (see section 5.2), we calculate all coupling metrics also at this level. Therefore we use the Rascal annotation relation m3@containment to relate all attributes to the compilation unit level.

CBO* (Coupling Between Objects count) metric:

First we calculate the number of method invocations of this class to other classes. Secondly we calculate the number of field accesses of this class to other classes. Subsequently we add both together. For the former calculation we use the annotation relation m3@methodInvocation but only for the methods, which excludes the constructor methods. For the latter we use the annotation relation m3@fieldAccess.
So we exclude the constructor invocations, because this implies new object creation and this metric excludes them. Also excluded are invocations to a super class. Therefore we exclude all relations from the annotation m3@extends.

CBO (Coupling Between Objects) metric:

This metric is in essence identical to the CBO* metric, but references from this class to other classes are only counted once. The CBO* relations are implemented as list relations, hence we store the CBO relations in a set relation, which excludes duplicates.

DAC (Data Abstraction Coupling) metric:

Since this metric counts the number of types used in this class from other classes we use the annotation m3@typeDependency but we exclude the primitive types and type variables.

ATFD (Access To Foreign Data):

This metric calculates the number of fields this class can access from other classes. We use the annotation m3@fieldAccess to retrieve these relations. Additionally this metric counts the number of accessor methods this class invokes from other classes.

We use the annotation m3@methodInvocation to retrieve these relations but only keep those methods that have a return type.

CA (Afferent Coupling):

This metric calculates the number of other classes that refer to this class. We use the inversion of metric CBO*.

8 Data Preprocessing Techniques Comparison and Result

Topics are assigned to documents[14] and documents have relationships with other documents, thus we can calculate the topic coupling. In section 8.1 we give definitions of inner and outer topic coupling. In section 8.2 we calculate the inner and outer topic coupling for the four data preprocessing techniques and present the results of the comparison.

8.1 Definition Inner and Outer Topic Coupling

Figure 8-1 depicts an abstract representation of the results of the chapters 6 Topic Generation and 7 Coupling Metrics Calculations. The designation with prefix "T" represents a topic and with prefix "D" represents a document. We denote the coupling relations between documents assigned to the same topic with Ti, where "i" stands for inner. The coupling relations between documents assigned to different topics we denote with To, where "o" stands for outer. The relations of Ti and To are disjunct for a given topic T.

For a formal definition of inner and outer coupling we customize the formula of Gethers and Poshyvanyk [6], which measures the probability between documents, to our needs.

Given a topic T the degree of inner coupling between the documents of this topic is defined as follows:

$$Ti_{k,m} = \frac{\sum_{x,y \in C_k} R_m(x,y)}{|c_k|}$$

$$where\ k \in \{0 \ldots K\}, m \in \{CBO, CBO*, DAC, ATFD, CA\}$$

and R_m is a function that returns the coupling relation between two documents for m and C_k is the set of all documents assigned to T_k.

We illustrate the calculation of Ti with an example based on figure 8-1.

[14] We call compilation units from this point on documents in accordance with the definition of Topic Modeling.

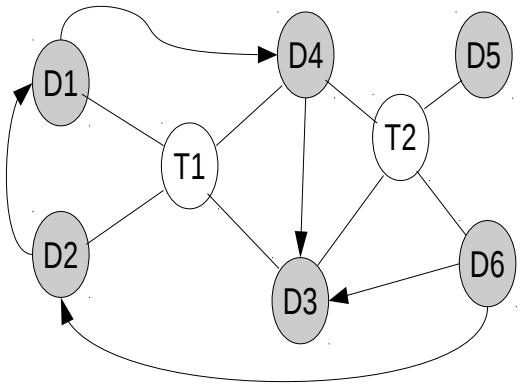

Figure 8-1: abstract representation of topic assignment and document coupling metric relations.

For topic T1 the coupling relation consists of the pairs (D2,D1), (D4,D3) and (D1,D4). For T1 the value of $|C_k|$ is 4 because the number of documents assigned to T1 is 4. As we count each coupling relations as 1, thus $R_m = 1$, the summation of R_m is 3. Therefore we get Ti is ¾ for topic T1. The Ti for topic T2 is 1/2.

For the degree of outer coupling between documents assigned to this topic and documents assigned to other topics we define:

$$\mathrm{To}_{k,m} = \frac{\sum_{x \in C_k,\, y \in (C - C_k)} R_m(x, y)}{|c_k|}$$

$$where\ k \in \{0 \ldots K\},\ z \in \{0 \ldots K\},\ m \in \{CBO, CBO^*, DAC, ATFD, CA\}$$

and R_m is a function that returns the coupling relation between two documents, the first assigned to T_k and the second to another T for m, C_k is the set of all documents assigned to T_k, C is the set of all documents.

We also illustrate the calculation of To with an example based on figure 8-1.
For topic T1 the coupling relation consists of the pairs (D1,D4) and (D4,D3) because both are a coupling relation in Ti of T1 the summation of R_m is zero. The number of documents assigned to T1 is 4, thus $|C_k|$ is 4, which leads to a To for T1 of zero (0/4). For Topic T2 we have the coupling relations (D6,D2) and (D6,D3). The latter is

excluded, due it is a relation in Ti for T2, therefore the summation of R_m is 1. The value for $|C_k|$ is 4, thus the To for T2 is 1/4.

8.2 Result

As depicted in figure 1-1, the process "Data preprocessing Techniques Comparison" is the final process and calculates the inner and outer coupling for the four data preprocessing techniques, which enables us to compare the four techniques with each other.

The input of this process are the different coupling metrics calculations and the document-topic matrix θ for each data preprocessing technique. The document-topic matrix θ is rotated to topic-document pairs, thus we know which documents are assigned to which topics. From the metrics calculation we know the coupling relationship between the documents.

The output of this process consists of five files per software system, representing the five coupling metrics, and each contains the values To and Ti for the four techniques. To visualize the result we produce box plots[15].

We consider high values for Ti better than low values because it shows high coupling between documents assigned to the shared topic. For To we like to see low values, considering it denotes less coupling between documents assigned to different topics.

For both software systems JHotDraw and jEdit we create box plots depicted in appendix B and C respectively. The bottom of the box denotes the 25% percentile, the top of the box the 75% percentile (successively lower and upper hinge), the horizontal line denotes the median and the horizontal dotted line the mean. To visualize the spread we use whiskers and outliers.

First we look at software system JHotDraw, see figure 12-5 till 12-14 in appendix B. The pattern we observe is that there is little difference between the data preprocessing techniques what the mean and median concerns and the values are relative equal spread between the whiskers. We find only a few outliers, but the upper hinges between the techniques differ the most. Taking all the differences and matches in account we conclude the four data preprocessing techniques look very similar to each other. Surprisingly the technique Raw which we added as an extreme technique per-

[15] https://plot.ly/

forms well in comparison to the others. For system jEdit, see figure 12-15 till 12-24 in appendix C, we observe a similar pattern as we see for JHotDraw.

We observe also another similarity. In almost all box plots the mean has a higher value than the median. This could mean that the data is skewed to the right which points to higher than smaller values. For Ti this means there are more topics with documents that have a high number of coupling relations. On the other hand we see the same for To. Which points to a high number of coupling relations between documents assigned to different topics.

9 Threats to Validity

As we spent a lot of effort in the experiment, we want to exclude unwanted conditions that could influence the result of the project. As depicted in figure 1-1 we have the processes "Verify result Metrics calculations with CKJM tool" not mentioned yet. In section 9.1 we describe this process.

We use as main programming language Rascal, which is an experimental language under development at the Centrum Wiskunde & Informatica (CWI)[16]. We have discovered a few bugs and created workarounds, but there is still a risk other bugs could influence the result. Section 9.2 covers the issues related to Rascal. The final section 9.3 in this chapter covers potential issues regarding the coupling metrics we use.

9.1 Verify result Metrics calculations with program CKJM

Process "Verify result Metrics calculations with CKJM tool" compares the result of the calculations for the metrics CBO and CA with the output of the program Chidamber and Kemerer Java Metrics (CKJM)[17]. CKJM calculates the metrics as defined by Chidamber and Kemerer [5,13] and additionally the metric CA [13]. CKJM is a command-line program and calculates the metrics based on the Java binary classes. The latter could be an issue because the metrics we calculate with Rascal are based on the source code. Another issue in comparing of software metric implementations is observed by Lincke et al. [14]. They found differences between different metric tools, including the program CKJM we use. Nevertheless we use the CKJM tool, but with caution.

We omit how to execute and use the program CKJM as we focus only on the differences between the two metric calculation outputs. We depict the results in the figures 9-1 till 9-4 with the use of box plots, as every dot represents a metric for a document. The tables 9-1 and 9-2 hold the figures of the box plots.

[16] http://www.cwi.nl/
[17] *http://www.spinellis.gr/sw/ckjm/*

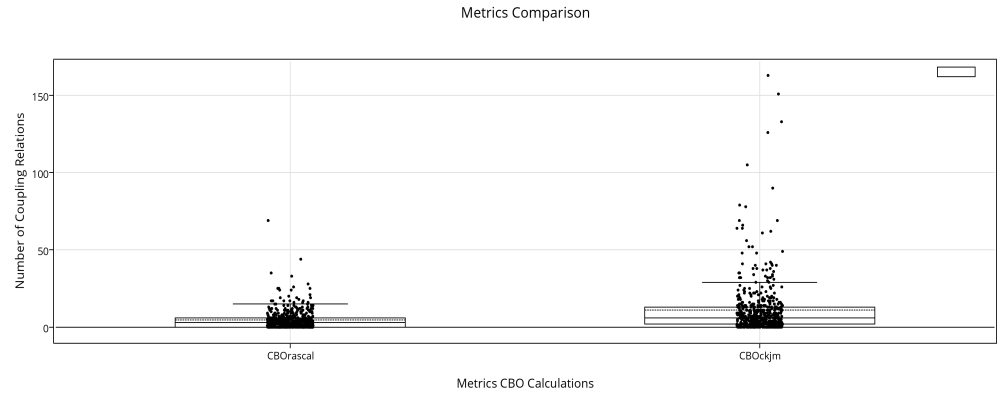

Figure 9-1: jEdit comparison metrics CBO Rascal and CKJM tool

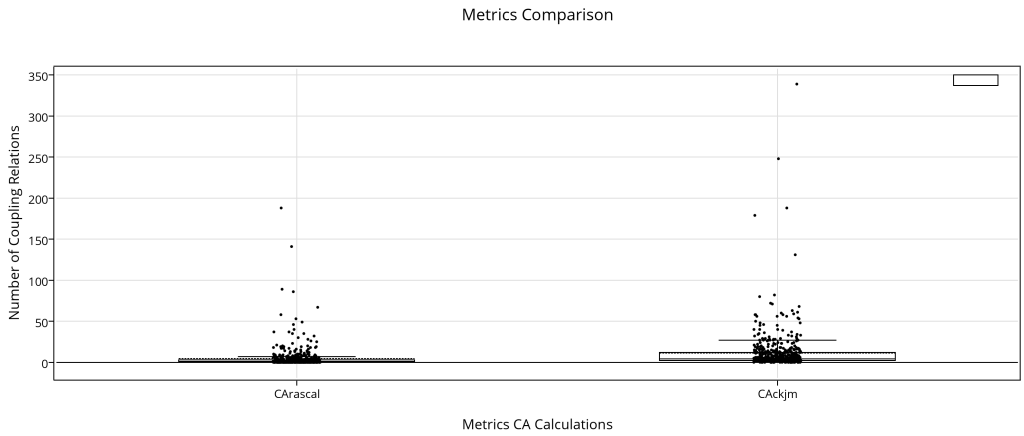

Figure 9-2: jEdit comparison metrics CA Rascal and CKJM tool

	CBO Rascal	***CBO CKJM***	***CA Rascal***	***CA CKJM***
Outliers	69	81	68	163
75% Percentile	6	13	3	8
Mean	4.5	7.6	3.4	11.1
Median	3	6	1	3
25 Percentile	0	2	0	1

Table 9-1: Box Plot Figures Metrics Comparison jEdit

For system jEdit, figure 9-1 and 2-1 and table 9-1, we see the values of the Rascal CBO calculation are about half of the CKJM calculation. As far as the CA calculations concern the Rascal calculations are about one third of the CKJM calculation.

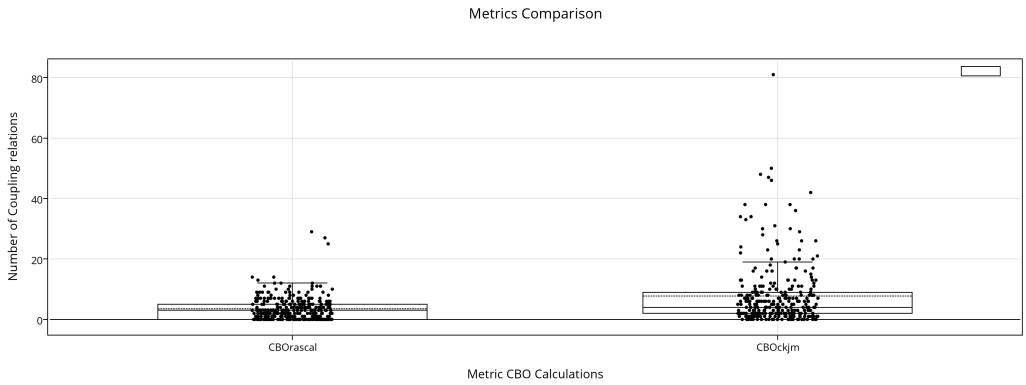

Figure 9-3: JHotDraw comparison metrics CBO Rascal and CKJM tool

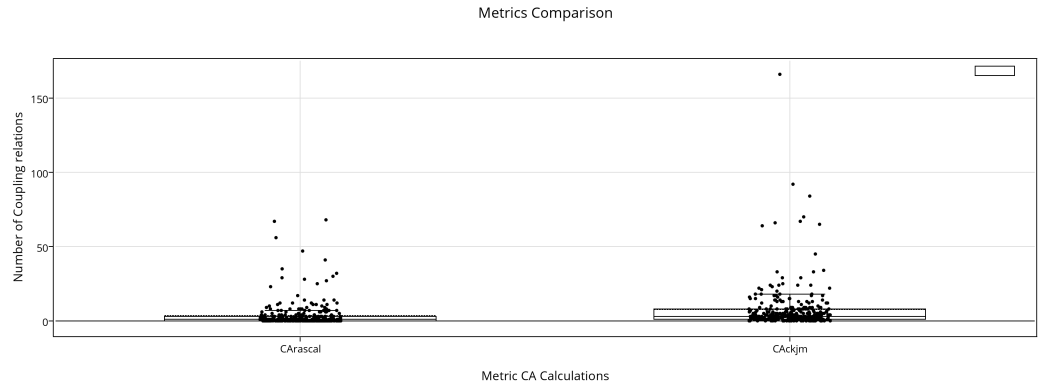

Figure 9-4: JHotDraw comparison metrics CA Rascal and CKJM tool

	CBO Rascal	***CBO CKJM***	***CA Rascal***	***CA CKJM***
Outliers	29	81	68	166
75% Percentile	5	9	3	8
Mean	3.4	7.6	3.4	7.6
Median	5	4	1	3
25 Percentile	0	2	0	1

Table 9-2: Box Plot Figures Metrics Comparison JHotDraw

The figures 9-3 and 9-4 and table 9-2 represent the metric values of JHotDraw, regarding the metrics CBO and CA, in comparison with the calculation of the program CKJM.

For JHotDraw we see the same pattern as for jEdit. The CBO metrics calculation is about half of the values of CKJM and one third of the CA metrics values. We see the

same difference or even at a larger magnitude in the research of Lincke et al. [14] We feel confident our metric calculation is trustful enough for our purpose.

9.2 Rascal issues

Considering Rascal is our main programming language for this research and Rascal is a research project at CWI which implies Rascal is not a mature programming language, Rascal could be a threat to our project. We have found several bugs and outline the workarounds:

a) Rascal supports reading and writing of CSV files. Unfortunately, when the file contains reals in E-notation, Rascal interprets these as a string value. When topic generation writes probabilities less than 0.5% in E-notation we have to exclude these documents from the topic assignments.
b) As we use M3 annotations for the calculation of metrics and the extraction of comments and documentation from the source code we highly depend on this model. Unfortunately, we found that the annotation M3@documentation does not contain the comments of fields, therefore we built a workaround and implemented the extraction of comments and documentation with the use of regular expressions.

9.3 Choice Coupling Metrics

We use five structural coupling metrics based on a study of De Souza and Maia [13], outlined in chapter 4 and section 9.3, as a measurement to compare the different data preprocessing techniques. The selection of the coupling metrics is based on the different kinds of coupling representations. In a study of Gethers and Poshyvanyk [6] 12 structural metrics are used in which three are the same as we use. These are the metrics CBO, DAC and CA. Gethers and Poshyvany mention in the threats to validity section that the structural metric they use could miss some kind of structural coupling in comparison with dynamic coupling metrics.

We conclude it is difficult to select the right set of coupling metrics and to be complete in covering all different kinds of couplings, nevertheless the result of our research depends greatly on them.

10 Related Work

Software repositories contain a wealth of valuable information about software projects [21]. In the past, these repositories were used only where they were intended for, such as maintaining versions of source code, tracking the status of defects [10] and archiving communication. Software products are developed to support operations on these repositories. For example, Subversion[18] is a popular tool in maintaining source code repositories.

Practitioners often depend on their experience, intuition and gut feelings [21] to make important decisions, but because software systems are getting increasingly more complex, larger in size and more systems are ecosystems [3], the need for more information from repositories and other sources to support the decision process increases. As the traditional tools and methods are not designed to provide this information, practitioners are getting more interested in the field of Mining Software Repositories (MSR) and IR. MSR analyzes and cross-links the data available in software repositories to uncover information from software systems and project [21]. In support of MSR, IR intends to utilize the unstructured and unlabeled text available in these software repositories [17,20]. Since this thesis is about topic modeling we focus on work in the field of the latter.

Grant et al. [8] successfully show how topic modeling can support the maintenance phase of software development. When a developer makes a change in the software, it is common that a significant part of the change is related to one topic. Grant et al. conclude that there is a relationship between latent topic models and co-maintenance history. The relationship can be used in the maintenance phase to prevent errors and suggest related changes.

In ongoing research Lopez [3] applies topic modeling to understand the evolution of software ecosystems. The work is intended to explore the role of topic location techniques in understanding the evolution of a software ecosystem and discovering patterns of topics evolving in the evolution of the software system.

Neuhaus and Zimmerman [18] apply topic modeling to the Common Vulnerability and Exposure (CVE) database. Although no source code is analyzed, it provides the developer useful information about how to avoid/eliminate vulnerabilities. For exam-

[18] http://subversion.tigris.org/

ple, Neuhaus et al. found that the majority of all vulnerabilities is caused by cross-side scripting, SQL injection, buffer overflow and not applying security updates to PHP. The use of topic modeling in this context is to find trends and it enables developers to stay ahead of others who want to exploit these vulnerabilities.

Gethers and Poshyvanyk [6] use topic modeling to discover new dimensions of coupling, which are not covered by structural coupling metrics. Their metric Relational Topic based Coupling (RTC) makes use of Relational Topic Modeling (RTM) which is an extension of LDA. As LDA discovers latent relationships between documents, RTM is also capable of predicting links between documents in the corpus.

Thomas et al. [20] introduce topic modeling to study the evolution of a software system and follow topics over time. They find that topics expose peak and drops that response to the actual change activity in the source code which makes evolution of topics observable and quantifiable.

The commonality between these studies, except the research of Neuhaus and Zimmerman, is that the researchers have to extract the unstructured data from the source in some way. As the extraction of the unstructured data is also used as the starting point of this research, it is also the starting point for these studies and has influence on the results. Gethers and Poshyvanyk uses a technique what we see as a mix of the Thomas and Selective technique. The research of Lopez is still in the exploration phase and no technique is defined yet. The others are described in this thesis elsewhere. We find another confirmation that these studies conduct a technique without reference to other research, or reasoning why the data preprocessing technique is used and is suitable for the research.

We added the research of Neuhaus and Zimmerman to give an example, how topic modeling can give a contribution to software development in a way not directly related to source code.

11 Conclusion and Future Work

In this chapter we present our conclusion based on the results of the experiment and how this matches with the hypothesis. Section 11.1 covers this topic. In section 11.2 we propose future work in the field of software data preprocessing for topic modeling.

11.1 Conclusion

In this thesis we defined four data preprocessing techniques to extract unstructured data from source code and built a tool that generates topics based on four data preprocessing techniques. Subsequently we developed a tool that calculates coupling metrics for the two software systems jEdit and JHotDraw. We used the metrics as a measure to compare the coupling between the documents assigned to topics. As a result, we observed that all data preprocessing techniques look similar to each other. In the hypothesis we predicted the technique Selective would result to a better topic distribution compared to the other three techniques. Based on the result of the experiment we have to conclude the hypothesis is neither wrong nor right.

Considering the results of the data preprocessing techniques look similar to each other we look to the similarities between the techniques. The techniques Grant, Thomas and Selective separate the programmer-defined names in components. Technique Grant also keeps the programmer-defined names in the vocabulary, but when we look at figure 12-2 in appendix A we see no programmer-defined words in the top-words (first 10 terms, defined as parameter in chapter 6 Topic Generation, with the highest probability of topic z, see section 3.1 Definitions of IR models) which could be decomposed into components. Continuing the comparison between these three techniques we see not that many Java keywords and programmer-defined names in the top-words of technique Grant. For example, the terms “int”, “the” and “class” are present in several topics of technique Grant because no stop-word list is used. We also used no stemming for the technique Grant but despite this, we see not that many terms that could be stemmed.

Figure 12-4 in appendix A depicts the top-words of technique Raw. We see in many topics programmer-defined names, as opposed to the techniques Grant. Notable is that the terms for JHotDraw are shorter than the terms for jEdit. We can only conclude that this is a system characteristic. Also notable is that the topics contain components as

terms together with the programmer-defined names themselves, despite we did not separate the programmer-defined name into components for technique Raw. For example topic 4 of jEdit contains the terms “line”, “firstline” and “physicalline” and for topic 8 the terms “plugin”, “jar” and “pluginjar”.
With the information we obtain from the experiment we cannot make a comparison between the techniques Grant, Thomas and Selective and the technique Raw.

11.2 Future work

In the previous section we made a comparison between the data preprocessing techniques. Although we can explain the similar results between the techniques Grant, Thomas and Selective, we cannot find a similarity between these three and the Raw technique.

The techniques Thomas and Selective are the most similar to each other, therefore we propose to merge them and replace the common English stop-word list by a customized one for each software system. We are in the opinion that unstructured data of a software system is different from regular unstructured data, for example from a novel, therefore we propose to experiment with a customized stop-word list such as proposed by Makrehchi and Kamel [4]. Regarding the extraction of unstructured data from source code we are in the opinion that technique Selective is the more appropriate one, in particular when we take a customized stop-word list into account. We can argue this with a question: why first take all text from the source code and remove it partly later, instead of just take what is needed?

As we mention in section 9.3 the completeness of the coupling metrics could influence the result. To avoid this uncertainty we propose to replace the coupling metrics with a vector representation of the relationships between documents and involve the topic to document assignment probability. For example Grant et al. [7] use such an approach.

Also, we propose to expand the experiment to other software systems with more documents. If the extended experiment confirms this preliminary result it means a significant reduction of effort using topic modeling for software systems.

12 Appendices

Appendix A: Top-words per technique

0 file vfs browser directory path filter method favorite view	0 figure draw event remove add change basic invalidate area
1 code component constraint grid extend row object return layout	1 figure locator south north east west relative bound align
2 task run keymap thread runnable manager update listener request	2 point double line length angle relative ctr len seg
3 action edit macro method view bean shell handler set	3 file project open save option pane sheet chooser change
4 class method object constructor static field args instance generate	4 expect number path text color stroke transform control found
5 color size font layout width row maximum minimum button	5 point curve param bezier error fit digitize hat path
6 entry table row model method column index set filter	6 figure bound handle task set restore transform basic lead
7 interpreter call stack node bsh error eval type callstack	7 path node bezier point index control segment chop set
8 abbrev map abbrevs completion word index keyword method key	8 flavor data object double return quad bound transfer support
9 node style tree result search string syntax hyper nod	9 view draw handle tool selection listener event select editor
10 option pane group label model general add save init	10 element attribute param return child namespace null full xml
11 menu context png param item edit dialog action property	11 layout row inset component number vertical column container leave
12 path vfs file param session error comp copy target	12 reader stream read system param data exception return current
13 key event shortcut bind input prefix handler binding modifier	13 return param method set point bound give visible change
14 method namespace space bsh variable interpreter script object import	14 method invoke param exist return obj default accessible getter
15 view widget status update caret method show font memory	15 width height frame border size dimension inset image paint
16 mouse menu action evt method update handler popup change	16 action label bundle resource util org jhotdraw app evt
17 dockable window dock layout entry method view bottom position	17 parser builder validator exception ixml xml create data throw
18 color gutter highlight text area paint font extension line	18 child figure composite layout layouter presentation bound double inset
19 token xsp statement literal expression jjtc jjtn bsh jjtree	19 attribute key figure set draw method color entry default
20 search replace match matcher start set pattern find case	20 class enum factory storable dom create base argument add
21 buffer view edit set pane marker change close method	21 figure mouse create evt tool editor draw prototype press
22 message log error source change debug send handler component	22 dom input output svg read write figure child attribute
23 history model set text url max method entry size	23 stroke grow color line width triangle chop fill diamond
24 line buffer fold edit start block end column text	24 edit undo redo presentation add composite return progress manager
25 method param return set edit add string link call	25 handle figure track anchor modifier lead step draw start
26 method chunk mirror xml list element handler tag start	26 exception line param error xml data occur system number
27 text register area data edit flavor transferable casanova list	27 menu bar action palette window item default osx tool
28 line screen physical offset visible scroll param text start	28 namespace prefix uri process method ixml associate exception null
29 font open icon selector property server firewall set proxy	29 element param attribute system xml line data throw exception
30 type primitive cast descriptor class null java wrapper object	30 icon property descriptor gen bean return default event array
31 class path loader source manager bsh map listener package	31 draw gen data applet set method init panel form
32 item class constant type insn pool link param writer	32 element current dom attribute document add tag node object
33 word param character tab char case whitespace line size	33 reader entity resolver xml ixml util read param str
34 array vector byte dimension length string put gap base	34 action property listener enable change set evt state event
35 visit label size instruction insn stack block param byte	35 sheet listener message param parent display component dialog type
36 selection caret line method text edit offset select end	36 writer xml print write string element indent pretty enm
37 plugin jar class load plugins resource path manager edit	37 prefs window preference float component frame toolbar handler run
38 indent edit line undo buffer bracket listener replace index	38 element attribute xml string param java map entity char
39 mode property file save load prop marker reload check	39 text holder size font bound inset draw layout tab
40 path file backup string return directory char param edit	40 draw editor view action create selection constrainer composite instance
41 encode print stream read line buffer detector page reader	41 draw editor button label tool create util resource bundle
42 bar split tool config active pane edit screen show	42 listener event notify interest register fire null create list
43 rule token context set parser line match end regexp	43 figure connector connection start target end find draw set
44 action button handler dialog list method class select panel	44 project application file create recent model app action init

Figure 12-1: Top-Words Selective technique. Left jEdit, right JHotDraw

0 stream encoding out exception write input read log reader	0 sheet the pane option message listener component parent owner
1 text register get history data flavor url area transferable	1 evt mouse get event handle void popup public tracker
2 line caret get selection offset start end buffer int	2 owner get point new anchor handle locator figure public
3 class name path get string loader source new classes	3 property value change action listener enabled get project void
4 edit property set get selected options new boolean undo	4 button editor stroke bar popup new add tool action
5 key event evt input get focus code stroke prefix	5 get new result drawing set string data null parameter
6 edit get pane split bar config tool set screen	6 undo redo edit public cannot exception name this return
7 plugin get jar string edit path null name log	7 editor view get drawing figures edit action public group
8 callstack node interpreter eval error get bsh stack public	8 get window prefs component bounds screen name palette evt
9 name mode get string property value props null properties	9 key get the resource string bundle argument button base
10 name namespace variable get null bsh space parent set	10 figure child children presentation index basic bounds layout composite
11 active return case string literal old dfa move break	11 insets layout rows left top int right get size
12 add set new get action selected list panel box	12 buf next append number new value path tokenizer token
13 file path string get backup return name directory edit	13 the name element exception xml param line system string
14 row size component grid col layout constraints get int	14 frames bounds frame all scroll height entry set result
15 menu get action item view set void mouse evt	15 reader this xml read str util char exception throws
16 buffer view get edit set pane null new marker	16 the name element attribute namespace string null this param
17 table get entry row model column int set selection	17 double bounds rectangle point width height public this anchor
18 start line end buffer block get int offset column	18 figure get connector connection start target end set point
19 name class method object this get error interpreter clas	19 the public this returns void for and method param
20 type class the get return descriptor name array modifiers	20 figure drawing public void figures remove event invalidated area
21 string int append buf return char len length new	21 icon return the descriptor bean null property get event
22 color font get property style set edit gutter highlight	22 get public new return null int for this add
23 size new byte int length label stack this put	23 name element value the string current attribute dom tag
24 the val name value item put class new string	24 point double curve error bezier hat first new points
25 width height get size left dimension top insets right	25 string this the xml name attribute writer element value
26 dockable window entry get name position layout string docking	26 listener event listeners list fire that edit this notify
27 action name get keymap string shortcut set edit label	27 text bounds get holder edit layout font insets size
28 line physical get int screen lines first scroll visible	28 method the class name object error exception return value
29 pane option tree group listener model event get name	29 out get read write attribute dom add public void
30 word line text offset get chunk index print sep	30 key value attribute get attributes this null figure set
31 tag string name last macro handler element equals null	31 double point return int angle math the line new
32 log message run error task void thread runnable this	32 file project value get set chooser option void app
33 rule token match end null context pattern line rules	33 new drawing view editor button set scroll undo pane
34 true token return case jjtn jjtree xsp node scope	34 action get menu add new set name project property
35 edit new view get property system null static action	35 figure created get prototype drawing null mouse view evt
36 search replace get set view text matcher and find	36 labels get action bundle util resource focus app evt
37 name the menu comp get string png context edit	37 application project model new set void name public string
38 the param method return this public for and null	38 stroke float get width grow double path break case
39 line indent get int buffer offset index fold seg	39 path node bezier point get index double mask public
40 text area line get int selection start caret gfx	40 new add editor tool bar button labels action attributes
41 node tree path result get search new default mutable	41 handle selection handles drawing scale factor rendering repaint value
42 type value lhs primitive return new rhs case error	42 view tool listener drawing void the public set container
43 vfs path file browser get directory session files string	43 image height img width color top icon insets left
44 index model get abbrev int abbrevs size list filter	44 data flavor restore exception clipboard object flavors transferable contents

Figure 12-2: Top-Words Grant technique. Left jEdit, right JHotDraw

0 widget view status set text gjt org edit java
1 primitive cast null error object wrapper operation java kind
2 fold level line buffer handler edit print page tab
3 line block buffer column offset size tab index text
4 edit property set select option box check add component
5 shortcut keymap string context label key null manager bind
6 font color style text render syntax glyph string null
7 offset edit undo listener mgr insert gap length remove
8 encode backup stream read java exception reader buffer input
9 property mode marker set save string prop load edit
10 active string literal dfa move state kind cur nfa
11 vfs file browser path directory filter favorite view set
12 task thread run runnable error update log awt request
13 rule match null tag pattern parser set token string
14 menu item edit split bar add pane tool config
15 error string stream log exception file system source print
16 event action mouse evt popup listener set menu key
17 line physical screen scroll visible manager display count buffer
18 history model url text set max string size index
19 register text data flavor area transferable edit string log
20 method object error eval null interpreter variable space namespace
21 buffer view edit set pane null change org jedit
22 tree node result path abbrevs abbrev string search mutable
23 table model entry row column set index list selection
24 node callstack lgpl eval interpreter bsh provision file error
25 dockable window layout width height dock entry top bottom
26 key event evt input modifier prefix completion null code
27 path map loader null manager string bsh source base
28 caret line selection text offset area select buffer position
29 param method string java org gjt object jedit set
30 string xml mirror set log write element equal handler
31 area text color line gutter highlight selection set caret
32 size visit insn put code label stack item constant
33 file path vfs string directory session log exception param
34 row size grid component layout color col width font
35 search replace set text matcher find view buffer directory
36 action view macro edit handler set shell bean string
37 option pane tree group model path listener event object
38 token true jjtn jjtree node scope false xsp scanpos
39 plugin jar string edit null property cache log load
40 icon menu edit png screen string param property component
41 line offset token indent context index text bracket buffer
42 string word length append buf index param len character
43 add set button action list panel layout box select
44 message component listener add log edit bus label source

0 element string current attribute dom add tag document object
1 application project model app create set recent string org
2 button popup stroke add editor tool bar put color
3 draw string set applet data result text object null
4 edit undo redo undoable change presentation exception false javax
5 attribute key set null object draw color figure stroke
6 figure create null prototype mouse tool draw target edit
7 sheet option pane message listener component parent param show
8 layout row inset component vertical container number extension parent
9 window component prefs palette focus screen event put evt
10 color map put number buf append token tokenizer path
11 point handle view anchor draw cursor lead track edit
12 transform figure bound affine decorator change restore rectangle data
13 action menu add bar set item open null recent
14 svg attribute add dom figure write read element exception
15 figure child presentation composite bound layouter add remove null
16 icon descriptor null bean property color event gen set
17 path bezier node point index mask control size add
18 text bound font holder layout edit inset size set
19 point curve hat bezier error param thread path fit
20 action bundle label resource util string jhotdraw org key
21 attribute element null namespace child string param full attr
22 method object invoke exception string param obj target exist
23 draw view editor button undo set scroll pane grid
24 inset height width bound leave top image border bottom
25 figure draw remove event invalidate add listener area edit
26 exception element line param system attribute data string xml
27 connector connection figure target set point null connect find
28 stroke path width line flavor fill decoration grow radius
29 file project app save open set chooser null unsaved
30 reader entity xml util resolver string read system str
31 editor view draw action figure group edit selection select
32 locator relative bound north south east west rectangle add
33 listener event list null notify fire interest protect edit
34 reader read stream current system exception buffer url java
35 java awt jhotdraw org point figure create geom extend
36 property listener change action enable project null state update
37 frame pane desktop set scroll internal arrange arrangement size
38 view tool evt mouse event draw handle key tracker
39 point angle line math rectangle length relative ctr geom
40 add editor tool draw button action bar attribute label
41 handle draw figure selection invalidate add view render set
42 exception software xml param ixml parser alter java data
43 rectangle width height grow point figure bound chop stroke
44 string attribute xml writer element write key param java

Figure 12-3: Top-Words Thomas technique. Left jEdit, right JHotDraw

0 log error file catch string out new ioexception static
1 jedit getproperty new options menu jcheckbox getbooleanproperty is
2 entry table int row new model void col column
3 primitive return type new lhs case value rhs boolean
4 int line physicalline textarea offset log debug getdisplaymanager first
5 buffer editpane view jedit bufferset null void buffers scope
6 callstack interpreter node evalerror eval simplenode new namespace
7 color component the font int row code col size
8 plugin jedit string jar null path name this pluginjar
9 jedit public message since pre class name static gjt
10 int the item type name final index constant value
11 null context pattern token line rules parserrule end match
12 true token return case jjtn false boolean xsp final
13 font style color print jedit float chunks static syntaxstyle
14 task runnable void run thread public instance log progress
15 tag string equals out xml write attrs name null
16 active return case jjmovestringliteraldfa break old long curchar int
17 path vfs file browser string session jedit directory vfsfile
18 name action string jedit shortcut label actions actionset keymap
19 selection caret buffer int the line offset void jedit
20 buffer int line start startline offset this end jeditbuffer
21 type the class return static string public method int
22 name null namespace class this object new interpreter method
23 void view jedit event public gjt new org awt
24 the method this param public return for and that
25 width insets height dimension int left color top getpreferredsize
26 search cons new jedit replace find searchandreplace start componer
27 encoding read url java history ioexception throws result historymode
28 new add jedit getproperty box borderlayout jpanel actionhandler void
29 keyevent evt key textarea null register text void action
30 the mode name jedit this log method since value
31 node defaultmutabletreenode new treepath tree path resulttree evt ir
32 jedit view gutter new getproperty jcheckbox options getcolorproperty
33 name class string path object optiongroup pane new null
34 int string text param append length return the char
35 entry string dockable name window null jedit view docking
36 path file jedit string name static buffer null value
37 textarea int gfx the public physicalline screenline void gutter
38 int chunk next return retval text end owner linetext
39 view buffer null jedit string static path the beanshell
40 name the png jedit param comp string static put
41 the lgpl public file under your provisions this and
42 jedit getproperty abbrev buffer mode string options abbrevs new
43 return new int string java public this org null
44 the this byte size length opcode label put final

0 index out bezierpath path void node throws bezierfigure get
1 view void handle tool drawing drawingview figure invalidatedarea editor
2 the this param exception string software not public xml
3 project file app value null final setenabled org evt
4 new editor put add bar labels color attr taskfigure
5 textholder text bounds editwidget gettext float layout insets figure
6 new add getaction model app project org putaction action
7 key thread value return string argument button null the
8 int rows insets layout cols number container parent allframes
9 this string the writer name xml write see param
10 point new double owner figure handle anchor public handles
11 public new return null void this for int private
12 the public this and void for returns method specified
13 new result the string null data getparameter domi getdrawing
14 public import jhotdraw java org class awt this return
15 the method object return new string methodname obj catch
16 point double that param points first new error last
17 new editor add labels rbmi toolbarbuttonfactory org jhotdraw attributes
18 return null the private static gen icon beandescriptor bean
19 the name namespace null string element param attribute child
20 key this object null attributekey value figure newvalue attributes
21 import public java jhotdraw labels org awt app key
22 methodname evt jtoolbar event awt java popupmenu jsobject savebutton
23 reader this xmlutil str read new the null throws
24 double float bounds rectangle result intersect entry get path
25 double point owner the from figure angle geom locator
26 editor view drawingeditor figures getview new group figure labels
27 double point int return the node mask line public
28 window prefs toolbar palette name screenbounds preferences bounds scree
29 new color map put buf number append value nexttoken
30 project application newvalue void action app oldvalue null propertychangelis
31 insets left top width height right bottom ddouble rectangle
32 stroke case break color static float get new final
33 figure child figures double drawing public children void point
34 null method target class submenu button menu else proxy
35 event listeners this void listenerlist that null for the
36 figure null connection connector target point start connectionfigure double
37 the param element string linenr systemid entity throws name
38 new view editor the undo newvalue org project pbutton
39 undo owner redo this public that return anedit super
40 the sheet message listener param parentcomponent owner pane dialog
41 double rectangle height width point grow bounds this anchor
42 flavor the data dataflavor clipboard requested not transferable xmltransferat
43 the current string name element value public document int
44 evt createdfigure new void figure null getview public mouseevent

Figure 12-4: Top-Words Raw technique. Left jEdit, right JHotDraw

Appendix B: Box plots JHotDraw

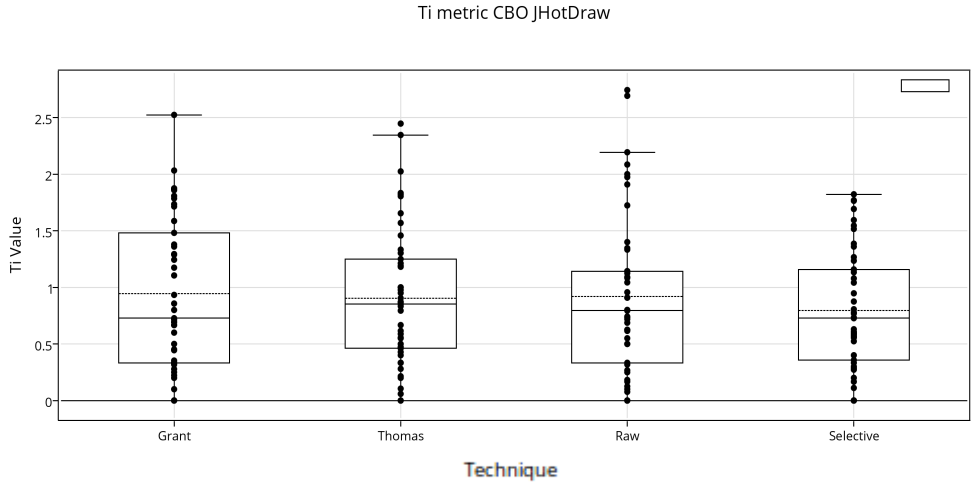

Figure 12-5: Box Plot Ti metric with CBO for JHotDraw

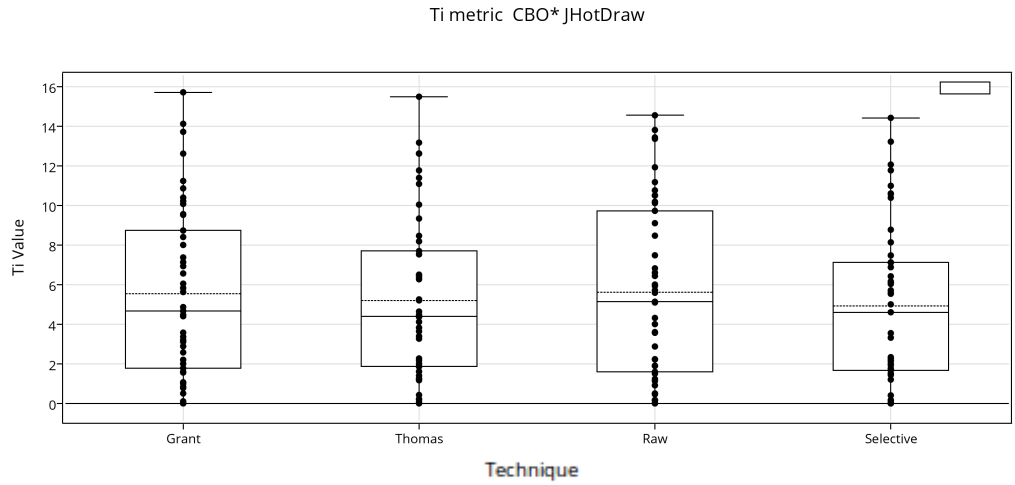

Figure 12-6: Box Plot Ti metric with CBO for JHotDraw*

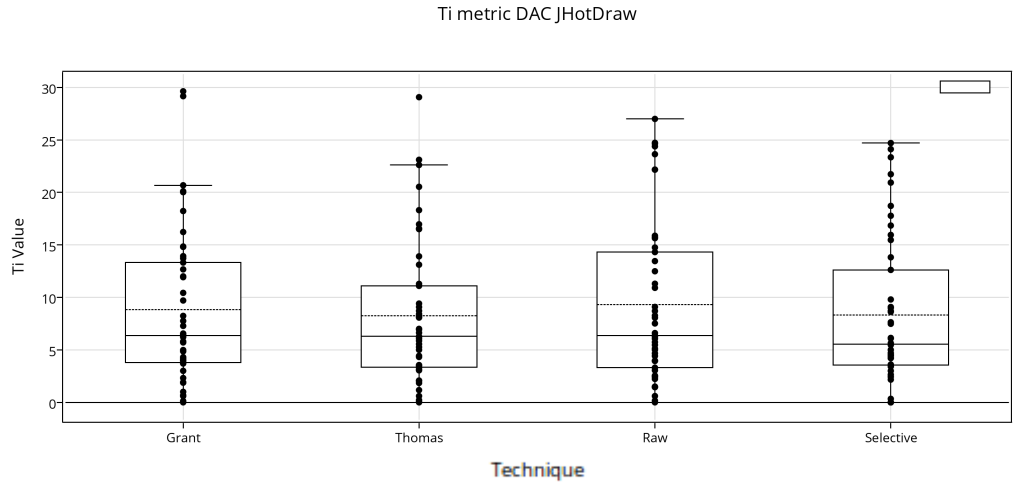

Figure 12-7: Box Plot Ti metric with DAC for JHotDraw

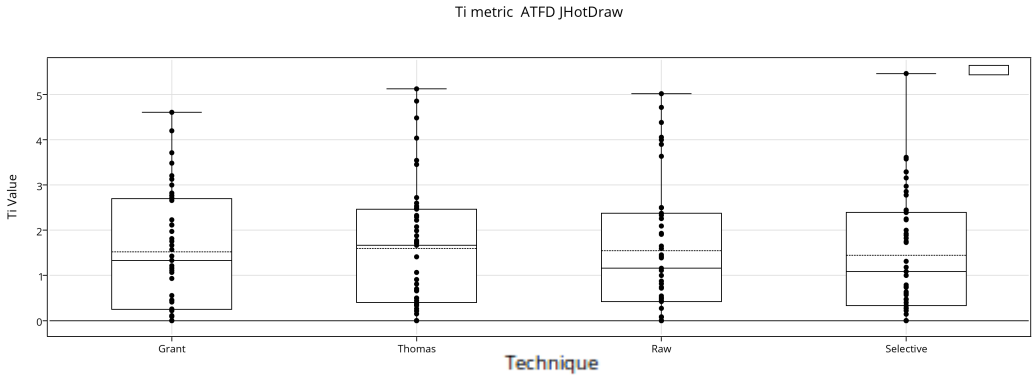

Figure 12-8: Box Plot Ti metric with ATFD for JHotDraw

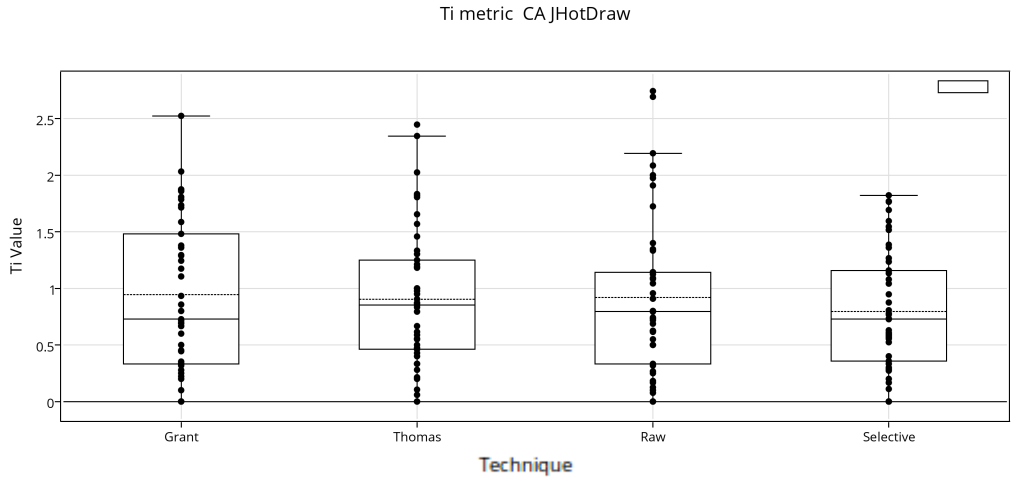

Figure 12-9: Box Plot Ti metric with CA for JHotDraw

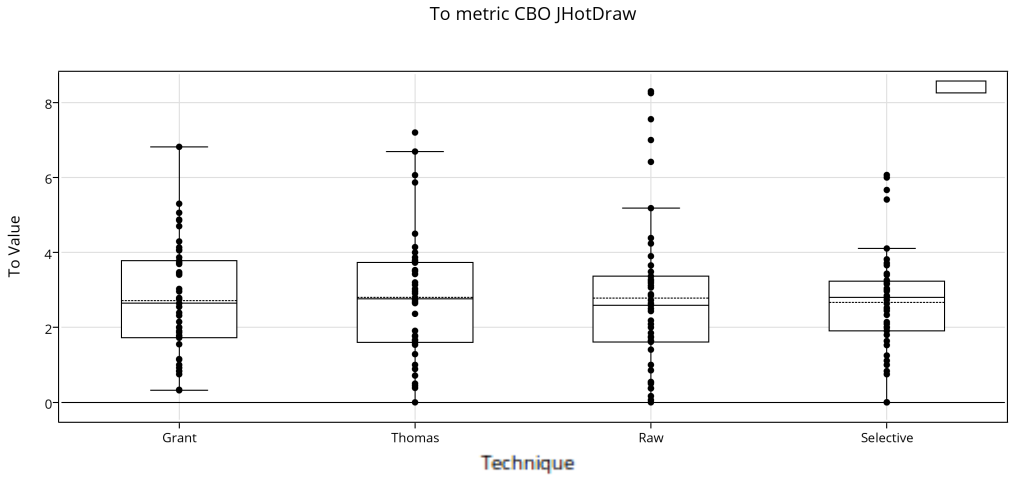

Figure 12-10: Box Plot To metric with CBO for JHotDraw

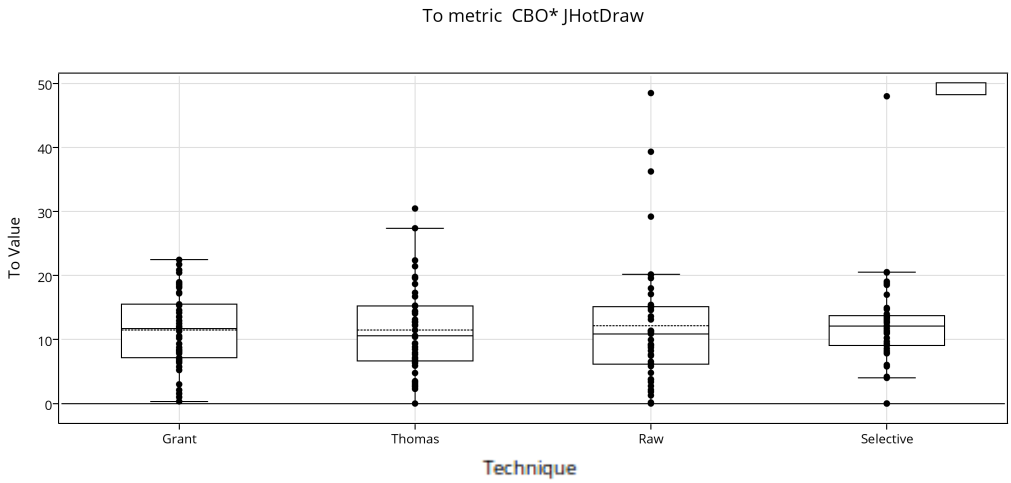

Figure 12-11: Box Plot To metric with CBO* for JHotDraw

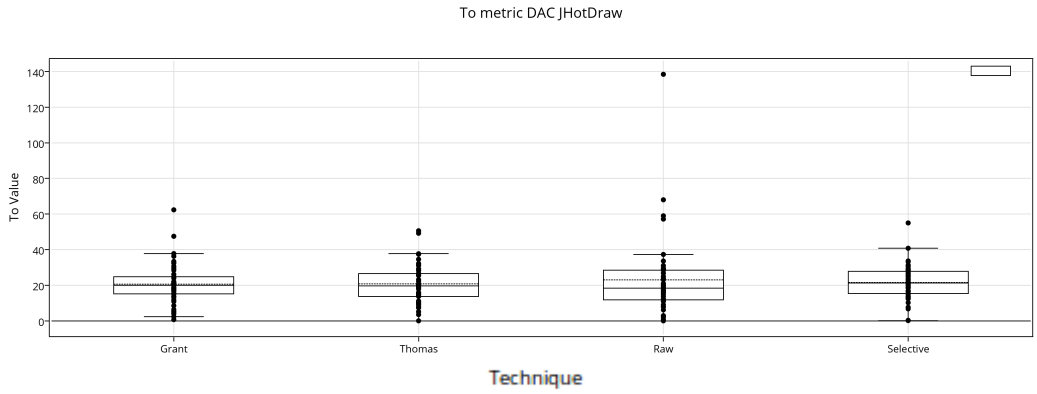

Figure 12-12: Box Plot To metric with DAC for JHotDraw

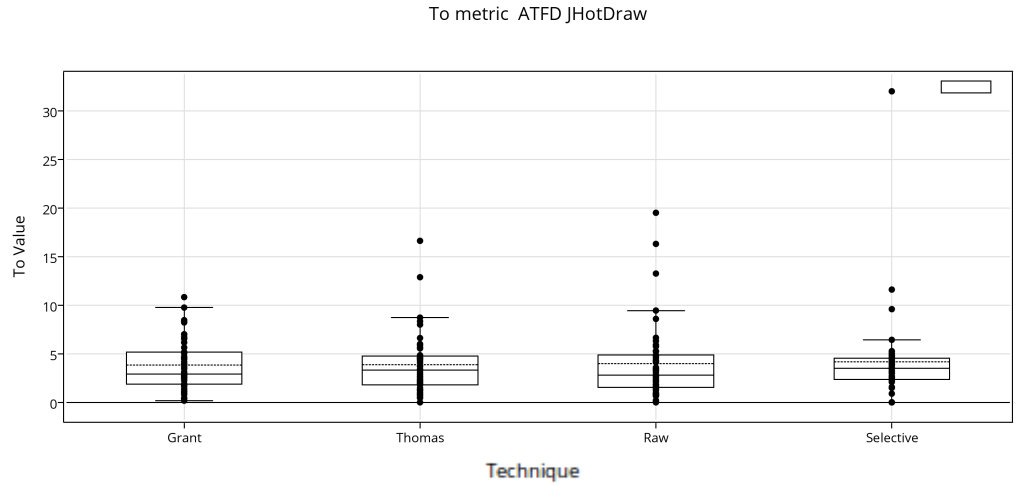

Figure 12-13: Box Plot To metric with ATFD for JHotDraw

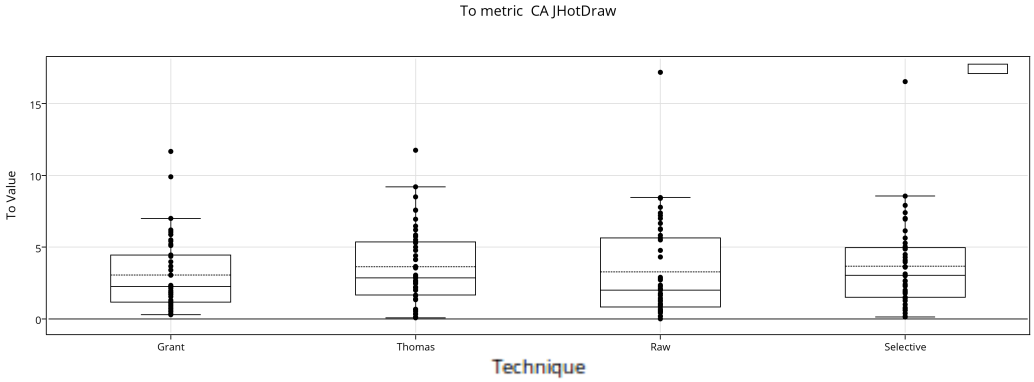

Figure 12-14: Box Plot To metric with CA for JHotDraw

Appendix C: Box plots jEdit

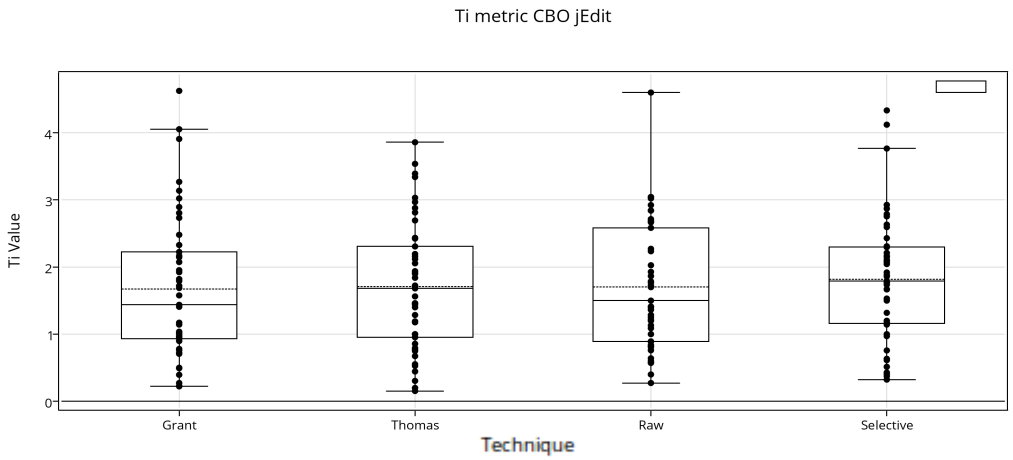

Figure 12-15: Box Plot Ti metric with CBO for jEdit

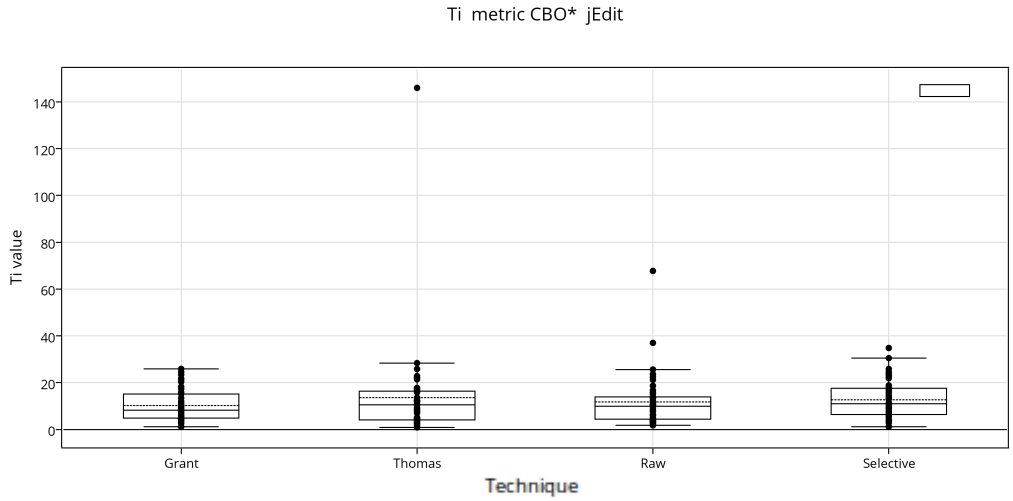

Figure 12-16: Box Plot Ti metric with CBO* for jEdit

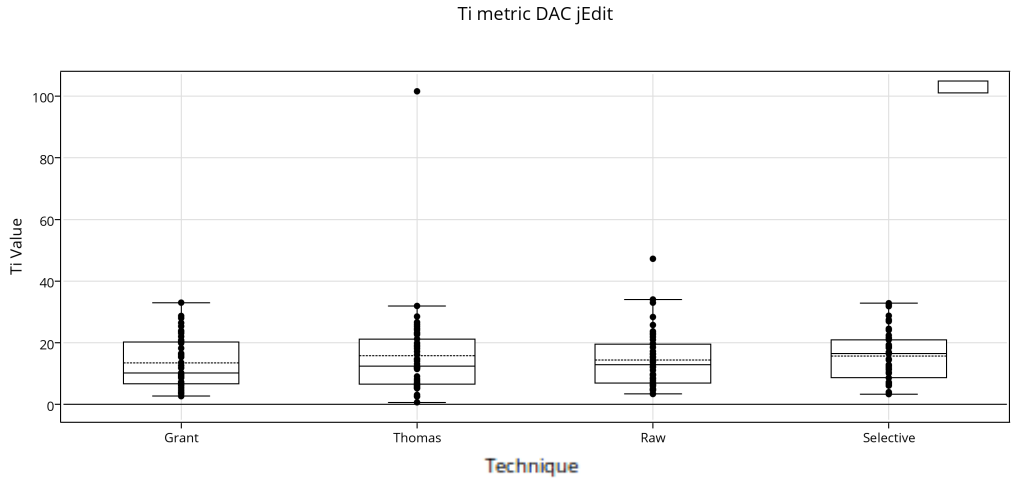

Figure 12-17: Box Plot Ti metric with DAC for jEdit

10

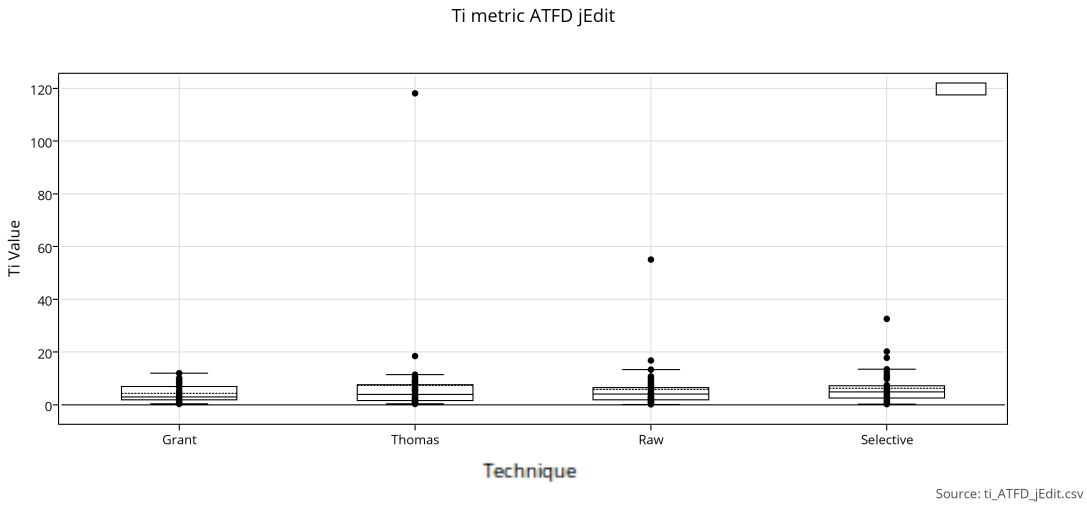

Figure 12-18: Box Plot Ti metric with ATFD for jEdit

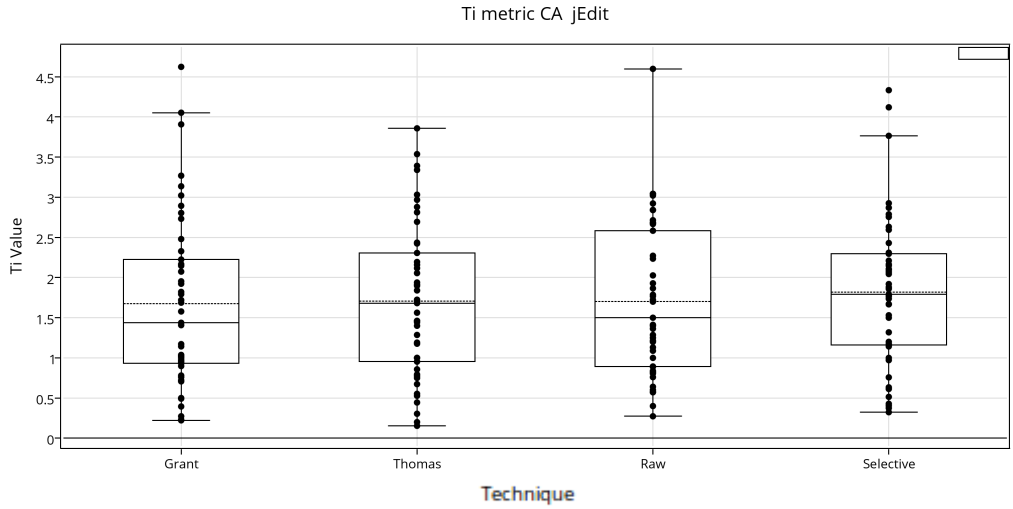

Figure 12-19: Box Plot Ti metric with CA for jEdit

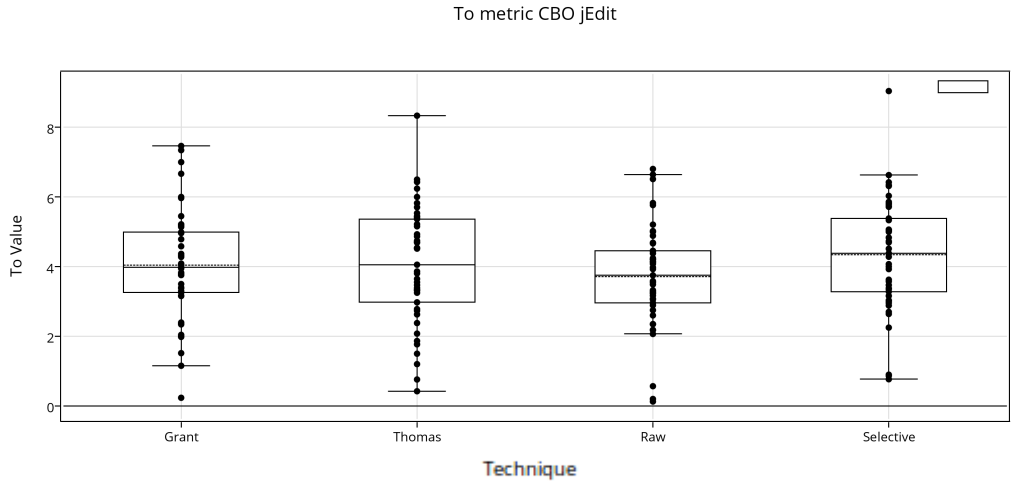

Figure 12-20: Box Plot To metric with CBO for jEdit

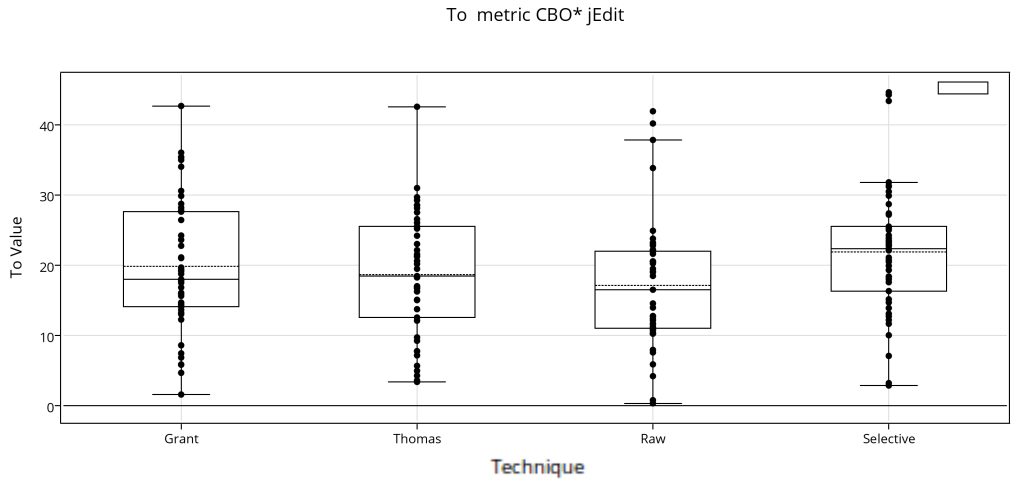

Figure 12-21: Box Plot To metric with CBO for jEdit*

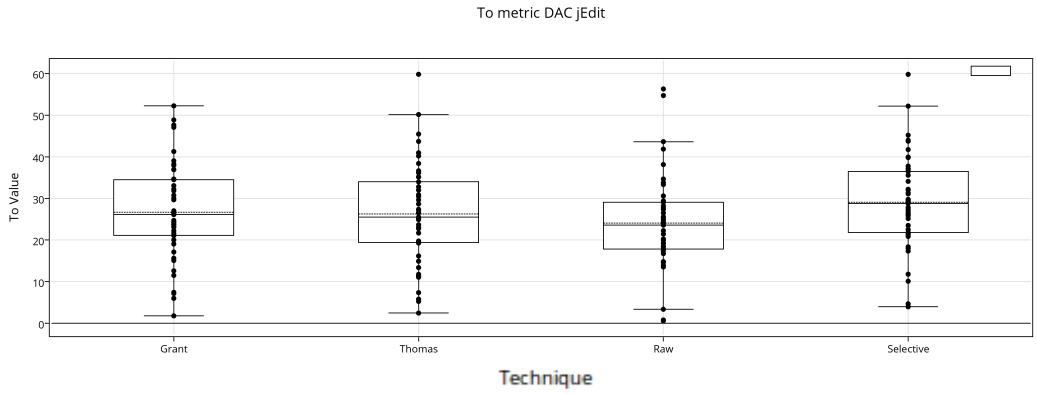

Figure 12-22: Box Plot To metric with DAC for jEdit

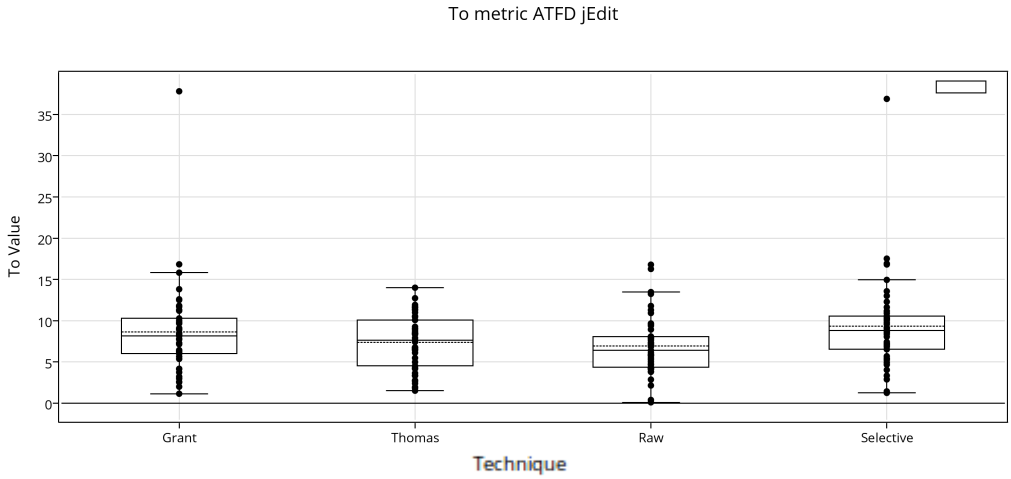

Figure 12-23: Box Plot To metric with ATFD for jEdit

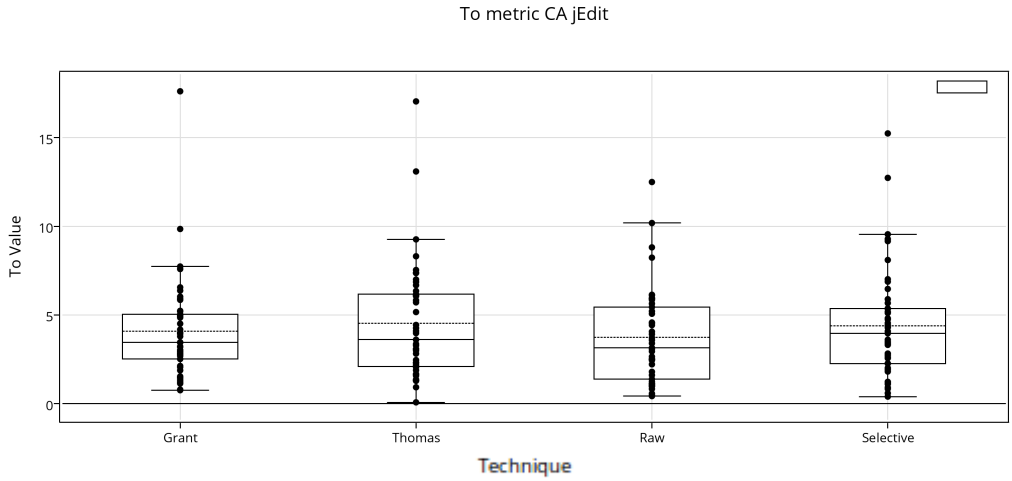

Figure 12-24:Box Plot To metric with CA for jEdit

13 Bibliography

[1] Blei, D.M. 2012. Probabilistic Topic Models. *Commun. ACM*. 55, 4 (Apr. 2012), 77–84.

[2] Blei, D.M., Ng, A.Y., Jordan, M.I. and Lafferty, J. 2003. Latent dirichlet allocation. *Journal of Machine Learning Research*. 3, (2003), 2003.

[3] Lopez, N. 2013. Using Topic Models to Understand the Evolution of a Software Ecosystem. *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering* (New York, NY, USA, 2013), 723–726.

[4] Makrehchi, M. and Kamel, M.S. 2008. Automatic Extraction of Domain-specific Stopwords from Labeled Documents. *Proceedings of the IR Research, 30th European Conference on Advances in Information Retrieval* (Berlin, Heidelberg, 2008), 222–233.

[5] Darcy, D.P. and Kemerer, C.F. 2005. OO metrics in practice. *IEEE Software*. 22, 6 (2005), 17–19.

[6] Gethers, M. and Poshyvanyk, D. 2010. Using Relational Topic Models to Capture Coupling Among Classes in Object-oriented Software Systems. *Proceedings of the 2010 IEEE International Conference on Software Maintenance* (Washington, DC, USA, 2010), 1–10.

[7] Grant, S., Cordy, J.R. and Skillicorn, D.B. 2013. Using heuristics to estimate an appropriate number of latent topics in source code analysis. *Science of Computer Programming*. 78, 9 (Sep. 2013), 1663–1678.

[8] Grant, S., Cordy, J.R. and Skillicorn, D.B. 2012. Using Topic Models to Support Software Maintenance. *Proceedings of the 2012 16th European Conference on Software Maintenance and Reengineering* (Washington, DC, USA, 2012), 403–408.

[9] Harrison, R., Counsell, S.J. and Nithi, R.V. 1998. An Evaluation of the MOOD Set of Object-Oriented Software Metrics. *IEEE Trans. Softw. Eng.* 24, 6 (Jun. 1998), 491–496.

[10] Hassan, A.E., Mockus, A., Holt, R.C. and Johnson, P.M. 2005. Guest Editor's Introduction: Special Issue on Mining Software Repositories. *IEEE Transactions on Software Engineering*. 31, 6 (2005), 426–428.

[11] Izmaylova, A., Klint, P., Shahi, A. and Vinju, J. 2013. M3: An Open Model for Measuring Code Artifacts. *arXiv:1312.1188 [cs]*. (Dec. 2013).

[12] Hills, M., Klint, P. and Vinju, J.J. 2013. Meta-language Support for Type-Safe Access to External Resources. *Software Language Engineering*. K. Czarnecki and G. Hedin, eds. Springer Berlin Heidelberg. 372–391.

[13] De Souza, L.B.L. and Maia, M.D.A. 2013. Do software categories impact coupling metrics? *Proceedings of the 10th Working Conference on Mining Software Repositories* (Piscataway, NJ, USA, 2013), 217–220.

[14] Lincke, R., Lundberg, J. and Löwe, W. 2008. Comparing Software Metrics Tools. *Proceedings of the 2008 International Symposium on Software Testing and Analysis* (New York, NY, USA, 2008), 131–142.

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

[16] Klint, P., Storm, T. van der and Vinju, J. 2011. EASY Meta-programming with Rascal. *Generative and Transformational Techniques in Software Engineering III*. J.M. Fernandes, R. Lämmel, J. Visser, and J. Saraiva, eds. Springer Berlin Heidelberg. 222–289.

[17] Thomas, S.W. 2012. *Mining Unstructured Software Repositories Using IR Models*. Ph.D. Dissertation. Queen's University.

[18] Neuhaus, S. and Zimmermann, T. 2010. Security Trend Analysis with CVE Topic Models. *2010 IEEE 21st International Symposium on Software Reliability Engineering (ISSRE)* (Nov. 2010), 111–120.

[19] Linstead, E., Hughes, L., Lopes, C. and Baldi, P. 2009. Software analysis with unsupervised topic models. *NIPS Workshop on Application of Topic Models: Text and Beyond* (2009), 52.

[20] Thomas, S.W., Adams, B., Hassan, A.E. and Blostein, D. 2014. Studying Software Evolution Using Topic Models. *Sci. Comput. Program.* 80, (Feb. 2014), 457–479.

[21] Hassan, A.E. 2008. The road ahead for Mining Software Repositories. *Frontiers of Software Maintenance, 2008. FoSM 2008.* (Sep. 2008), 48–57.