UNIVERSIDADE FEDERAL DE CAMPINA GRANDE

CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA

COORDENAÇÃO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

LUCAS RANIÉRE JUVINO SANTOS

IMPROVING TRACEABILITY RECOVERY BETWEEN BUG REPORTS AND MANUAL TEST CASES

CAMPINA GRANDE - PB

2020

Universidade Federal de Campina Grande

Centro de Engenharia Elétrica e Informática

Coordenação de Pós-Graduação em Ciência da Computação

Improving Traceability Recovery Between Bug Reports and Manual Test Cases

Lucas Raniére Juvino Santos

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Computer Science

Linha de Pesquisa: Software Engineering

Franklin Ramalho and Tiago Massoni

(Supervisors)

Campina Grande, Paraíba, Brasil

S237i Santos, Lucas Raniére Juvino.
Improving traceability recovery between bug reports and manual test cases. / Lucas Raniére Juvino Santos. - Campina Grande, 2020.
78 f. : il. Color.

Dissertação (Mestrado em Ciência da Computação) - Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática, 2020.
"Orientação: Prof. Franklin Ramalho, Prof. Tiago Massoni".
Referências.

1. Engenharia de software. 2. Artefatos de software. 3. Computer science. 4. Software engineering. 5. Bug reports. I. Ramalho, Franklin. II. Massoni, Tiago. III. Título.

CDU 004.41(043)

FICHA CATALOGRÁFICA ELABORADA PELO BIBLIOTECÁRIO JESIEL FERREIRA GOMES CRB-15/256

IMPROVING TRACEABILITY RECOVERY BETWEEN BUG REPORTS AND MANUAL TEST CASES

LUCAS RANIÉRE JUVINO SANTOS

DISSERTAÇÃO APROVADA EM 30/10/2020

CAMPINA GRANDE - PB

Resumo

Durante o ciclo de vida de um software há a produção de diversos tipos de artefatos, responsáveis por documentar e organizar não apenas o software, mas também o processo de desenvolvimento. Muitos desses artefatos são codependentes, mas apesar da manutenção da consistência entre os artefatos ser uma tarefa importante, não é trivial. Já que são produzidos por indivíduos diferentes, em diferentes tipos de detalhes e linguagens, a automação é altamente desejável.

Algumas pesquisas investigam a rastreabilidade entre bug reports e casos de teste manuais. Uma vez que casos de teste são uma forma popular de documentar requisitos em projetos ágeis, esse tipo de rastreabilidade permite, por exemplo, analisar como bug reports se relacionam com os requisitos. Uma pesquisa anterior avaliou três técnicas de Recuperação de Informação (LSI, LDA, e BM25) e um algoritmo de Deep Learning (Word Vector) usadas para recuperar links entre bug reports e casos de teste. Os resultados dessa pesquisam apontam a necessidade de melhorias.

Em nossa pesquisa avaliamos um conjunto de técnicas de melhorias aplicadas em um trabalho base, que utiliza artefatos do projeto open-source Mozilla Firefox. As técnicas de melhorias são: (i) limpeza textual e de informação; (ii) correção ortográfica; (iii) aumento do peso de campos do bug report (duplicação do título e descrição); (iv) junção de matrizes de similaridade; e (v) junção de matrizes de rastreabilidade.

A avaliação das técnicas de melhoria é feita através da comparação entre os resultados obtidos por elas e os resultados obtidos pelo trabalho base, em termos de precisão e cobertura. Foi observado uma leve melhoria nas taxas de precisão e cobertura nas técnicas de limpeza textual e de informação, correção ortográfica, duplicação do título, e na combinação dessas três técnicas. Em uma das estratégias da técnica de junção de matrizes de rastreabilidade, que seleciona traces recuperados por ao menos uma das técnicas de rastreabilidade, alcançou um valor de cobertura de 93%.

Abstract

During the life cycle of a software there is the production of several types of artifacts, responsible for documenting and organizing not only the software, but also the development process. Many of these artifacts are codependent, but while maintaining consistency between artifacts is an important task, it is not trivial. Since they are produced by different individuals, in different types of details and languages, automation is highly desirable.

Some research investigates traceability between bug reports and manual test cases. Since test cases are a popular way of documenting requirements in agile projects, this type of traceability allows, for instance, to analyze how bug reports relate to requirements. Previous research evaluated three Information Retrieval (IR) techniques (LSI, LDA, and BM25) and a Deep Learning (DL) algorithm (Word Vector) used to retrieve links between bug reports and test cases. The results of this research point to the need for improvements.

In our research we evaluate a set of improvement techniques applied to a baseline work, which uses artifacts from the open-source Mozilla Firefox project. Improvement techniques are: (i) textual and information cleaning; (ii) spellchecking; (iii) increasing the weight of fields in the bug reports (duplication of title and description); (iv) merge the similarity matrices; and (v) merge the traceability matrices.

The evaluation of improvement techniques is made by comparing the results obtained by them and the results obtained by the baseline work, in terms of precision and recall. There was a slight improvement in the precision and recall rates in the techniques of textual and information cleaning, spellchecking, duplication of the title, and in the combination of these three techniques. In one of the strategies of the technique of merging the traceability matrices, which selects traces recovered by at least one of the traceability techniques, it reached a coverage value of 93%.

Agradecimentos

Primeiramente gostaria de agradecer aos meus orientadores, Franklin Ramalho e Tiago Massoni, pela paciência, orientação e ajuda durante meu caminho pelo mestrado. Mesmo não sendo o orientando mais comunicativo, sempre me senti incentivado ao diálogo, e sempre se mostraram prestativos.

Agradeço também aos meus pais, José Antônio e Merivane, pela dedicação, apoio e carinho recebidos não apenas durante o período do mestrado, mas em toda a vida. E por sempre entenderem quando eu dizia que não poderia viajar para visitá-los.

Aos meus amigos, tanto os de Campina Grande quanto os de Arapiraca, por sempre se manteram presentes nos momentos de cansaço e desânimo, mesmo a maioria estando distante.

Aos colegas e professores da UFCG, e a todos os funcionários que influenciaram, de forma direta ou não, na conclusão dessa fase em minha vida.

Contents

List of Symbols

IR - *Information Retrieval*

DL - *Deep Learning*

LSI - *Latent Semantic Indexing*

SVD - *Singular Value Decomposition*

LDA - *Latent Dirichlet Allocation*

BM25 - *Best Match 25*

ML - *Machine Learning*

NLP - *Natural Language Processing*

List of Figures

List of Tables

Chapter 1

Introduction

Software artifacts are by-products of the development process and help to describe architecture, functions, and design of the software [36]. Among these artifacts we can mention, as example: requirements documents, test cases, and bug reports. These artifacts are necessary for understanding and organizing the program and its development process, as they provide to the people involved in the process a common and well-structured information base.

However, it is not a trivial task to maintain the quality and consistency of those artifacts throughout the development and evolution of the program. Engineers working on large-scale software development have to manage a large amount of information, spread over several artifacts [8]. Because of this, traceability of artifacts has become a task of fundamental importance in software engineering.

1.1 Motivation and Context

Software artifact traceability is the ability to describe and follow the life of an artifact during the software life cycle [4]. The purpose of the traceability is to discover relationships between artifacts to facilitate the recovery of relevant information, which is necessary for a variety of engineering tasks [5], and can also be used to automate some activities of the software process.

The need for automatic techniques for recovering traceability between software artifacts increases as the complexity of the projects increases [5]. For this reason, tools [37; 24; 33] and approaches [16; 23; 2] that seek to study techniques for recovering relationships between various software artifacts have been proposed and evaluated, using several computational techniques to achieve this goal, such as Information Retrieval [8] (IR) and Deep Learning [19] (DL) algorithms.

1.2 Problem

Previous studies have researched the recovery of traceability between several types of textual software artifacts, however we noticed only a few studies *relating and tracing bug reports to manual test cases* [12; 23], despite test cases often being the most up-to-date documentation and naturally linked to bug reports. These studies did not obtain satisfactory results, in terms of *precision* and *recall*, which shows a deficiency for the practical use in software development teams.

The lack of studies relating those two artifacts plus the non-optimistic results, make us still do not have a clear idea of which techniques are the best to apply or how best to apply them in the context of recovering traceability between bug reports and manual test cases. This means that it is still an open topic for researching new approaches or attempts to improve existing ones.

A specific research report [12] evaluated three IR techniques (LSI, LDA, and BM25) and a DL algorithm (Word Vector) used to recover traceability links between bug reports and manual test cases using historical data from the Mozilla Firefox open-source project (this being our baseline work). Although the study reports one specific IR technique performs best (LSI) within certain experimental conditions, the results, in terms of precision, recall, and F_2-Score, are deficient for practical utilization by a software team

1.3 Problem Relevance

Traceability plays a important role in development process. With the access to trace information, the development and testing teams can perform work tasks such as impact analysis, requirements validations, and identification of reusable artifacts [8].

Bug reports and manual test cases are often the most up-to-date artifacts in software development, specially in agile development teams. In real projects the amount of these artifacts quickly reaches thousand. Despite that, the task of recovering the traceability between these artifacts remains fully manual, laborious, error-prone, and tedious work.

When a bug is reported there are some procedures performed by the test team. First, it is necessary to check the occurrence of the bug, as not all bug reports contain the steps taken, it is necessary to know what to test (system feature), and how to test (which steps are performed). One of the possibilities is to try to find manual test cases that can be used for that bug report, but this task can be costly and time consuming, because it is done manually.

The traceability recovery between bug report and manual test cases - in direction from bug reports (query) to manual test cases (target) - can bring some benefits for that process. For instance, it can allow to automatically recover some manual test cases that failed to find determined reported bug; or even relate a bug report to a determined group of test cases, making possible to add new test cases for the next versions of the software; in addition to the possibility of automatically recommending manual test cases, to facilitate the process of finding relevant test cases for a given bug report.

1.4 Objective

This research have as main goal mitigate the problems related to the traceability recovery between bug reports and manual test cases, by evaluating a set of improvement techniques (Chapter 3) applied on a specific work [12], that we use as a baseline work. The baseline work is explained in more detail later 2.5, but in summary, it is a study that uses three IR (LSI, LDA, and BM25) and one DL (Word Vector) techniques to recover the traceability between bug reports and test cases.

In addition to the main objective, we also aim to present a discussion about the artifacts used in the study and the existing limitations in the approach studied and the proposed improvement techniques.

1.5 Scope

The scope of the research is the field of Software Engineering, in the context of software artifacts traceability. Specifically, our research propose to study improvement techniques for a previous baseline work [12], a case study that uses IR (LSI, LDA and BM25) and DL (word Vector) techniques to recover the traceability between bug reports and test cases of a specific open-source project, the Mozilla Firefox. The improvement techniques being as follows:

1. Text and information cleaning;
2. Spellchecker;
3. Bug report fields weighting;
4. Similarity matrices merging;
5. Traceability matrices merging.

1.6 Contributions

In this dissertation, we apply and evaluate five improvement techniques on a baseline work [12]. The baseline work is an approach that uses IR and DL techniques to recover traceability links between a bug reports an manual test cases. The improvement techniques were defined after the analysis of the results of the base work, and were chosen to mitigate some possible failures of the baseline work, and are better detailed and defined later (Chapter 3). The improvement techniques covered by this research are:

1. Text and information bug report cleaning;
2. Bug report Spellchecking;
3. Bug report title and description weighting;
4. Similarity matrices merging;
5. Traceability matrices merging.

To evaluate the applied improvement techniques, we compare the results obtained by the baseline work with the results obtained after the application of the improvement techniques. The experiments are carried out and evaluated in two scenarios: (i) threshold value 0.0; and (ii) threshold 0.9.

We use *Precision*, *Recall*, F_2*-Score*, and *Goodness Scale* as evaluation metrics. For the qualitative analysis, we also compare changes related to the *Traces Recovered by All Techniques* (True Positives) and *Traces Missed by All Techniques* (False Negatives).

We believe these results take an essential step forward in the automation of traceability recovery between these artifacts in the software process, which has great potential to assist developers.

The dissertation has the following contributions:

- Application and evaluation of improvement techniques applied on a baseline work;
- Comparative study of the evaluated techniques;
- Discussion about bug reports in the context of traceability;

1.7 Document Organization

The dissertation is organized as follows. Chapter 2 presents basic concepts covered in the research and introduces the baseline work. In Chapter 3 we introduce the proposed improvement techniques. The Chapter 4 shows the methodology followed by the research. Chapter 5 presents and discusses the results. Chapter 6 discusses the related works, whereas Chapter 7 presents the final considerations.

Chapter 2

Background

This chapter presents concepts and fundamentals of topics addressed in research. Starting with the definition of bug reports, followed by the definitions of test cases. Then we define the IR (LSI, LDA and BM25) and DL techniques (Word Vector) used in the approach. Lastly, the baseline work is presented.

2.1 Bug Reports

Bug tracking systems play an important role in many software projects as they allow the communication between the users community and the developers [21]. This communication benefits both sides: for the users these systems become a means of exposing your experience while using the system program. For the developers the bug tracking systems provide details about several defects found by users while using the program. This communication is made thought bug reports.

Bug reports describe any software failure identified by the user, or automatically by the system itself [10]. The bug reports provide detailed information about the failure, helping the developers to investigate the cause and fixing the bug. A bug report is formed by many information fields, such as title, type, priority, product, system version and description, which may vary depending on the issue tracking system. The bug report description may contain several valuable information for the developers, for instance: reproduction steps, stack traces, code snippets, error messages, expected behavior and observed behavior.

In the Figure 2.1 is possible to see an example of a bug report, taken from the Mozilla

Bugzilla[1]. This example allows us to view some of the information fields present in Bugzilla bug reports and their respective contents. In this particular example, we can see what we might call a good quality bug report; the author wrote a very detailed description, including steps to reproduce, and the expected and obtained results.

During the presentation of this research, some bug reports will be named by their respective IDs, the bug reports can be checked by searching for their IDs on the Mozilla Bugzilla website.

2.2 Manual Test Cases

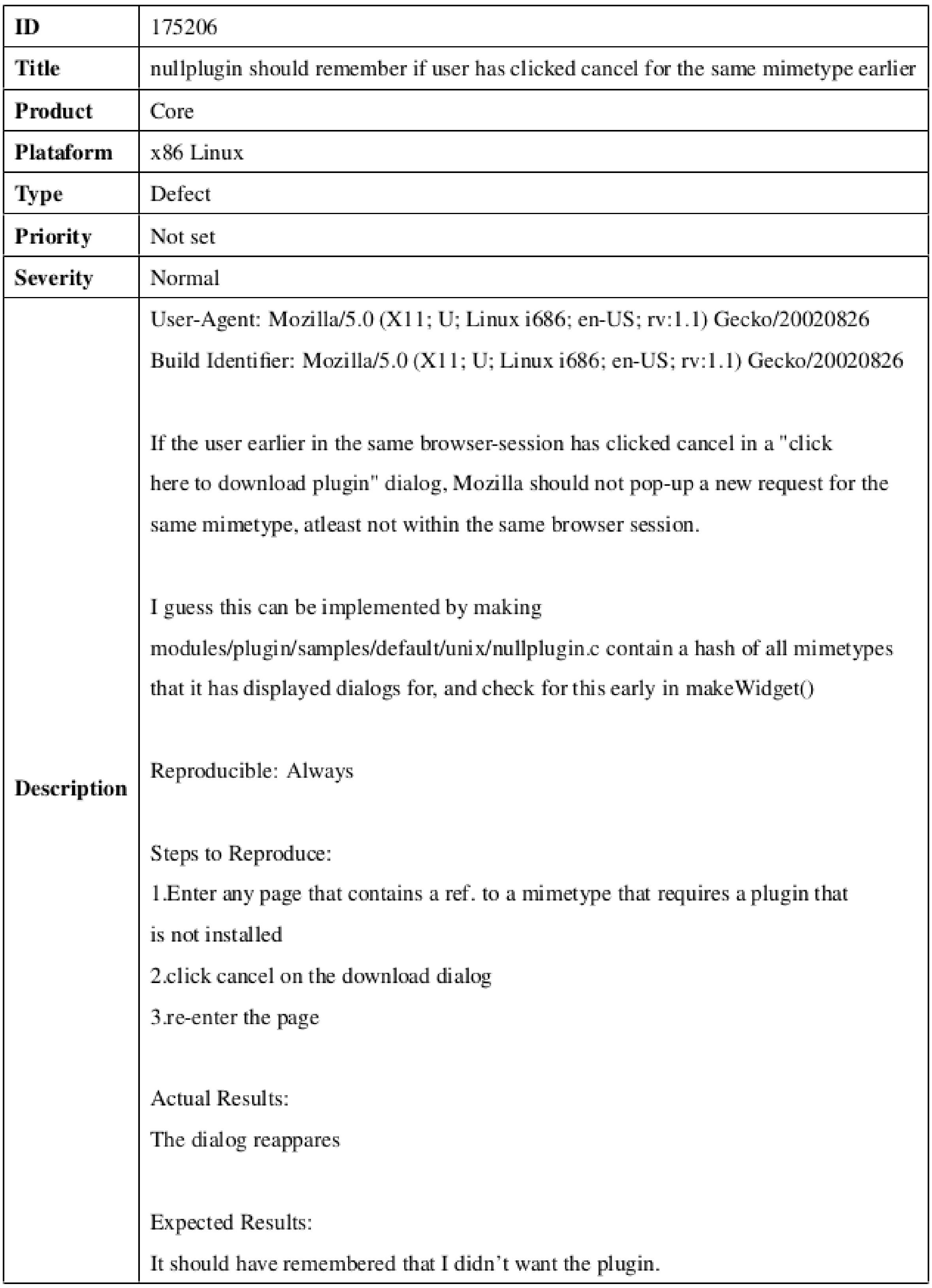
Software testing have a fundamental importance in the software development, evolution, and maintenance. Test cases allows to detect bugs before the software release. Can be described as a group of procedures executed to evaluate some aspects of a piece of software, by observing its execution [1]. The testing process can be divided into two main categories: manual and automated testing. Automated testing uses a software, different form the system being tested, to control the execution of the tests, and comparing the software outputs to a predefined one to ensure the correct behavior of the system. In manual testing, the tester uses manual test cases to interact with different functionality present in the system, simulating a user, to eliminate all possible errors [34].

Manual test cases define a textual sequence of steps, including the expected behavior, that allows a tester to determinate if a software is following the software requirements [36]. In Figure 2.2 is shown an example of manual test case taken from Mozilla Quality Assurance[2]. The *Title* is a short description of the test case purpose. The *Steps* field present the steps to reproduce, for each step is defined a expected result, present in *Expected Result* field. The result obtained by each step to reproduce must match its respective expected result, this is checked by the tester in the test process.

In the example we can see test case number 155, from test day 2016/10/28. The "Downloads Dropmaker" feature has been tested, and the steps followed by the testing team can be seen in "steps". It also determines the result expected for the feature to pass this specific test.

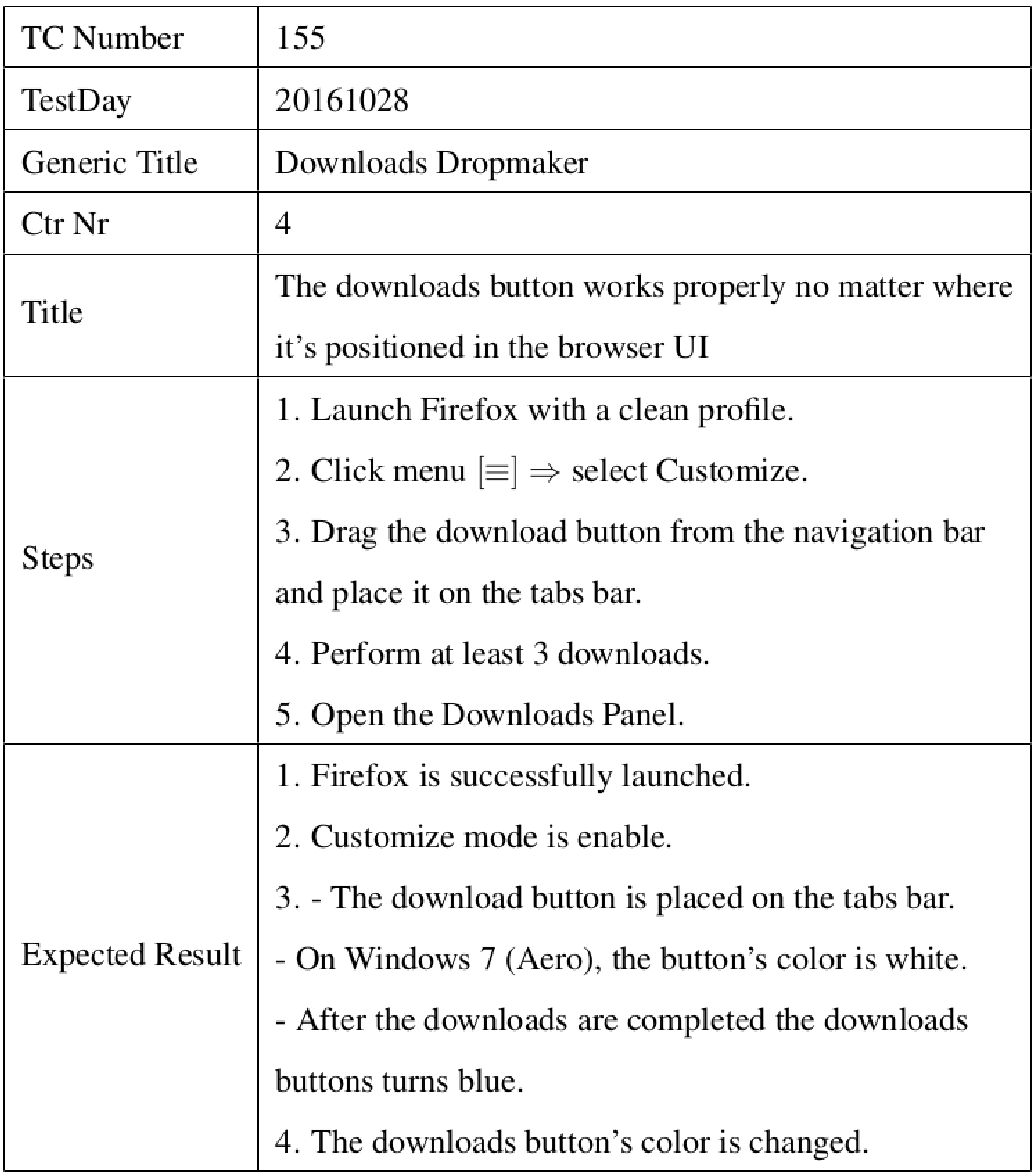
[1]https://bugzilla.mozilla.org/

[2]https://quality.mozilla.org/



ID	175206
Title	nullplugin should remember if user has clicked cancel for the same mimetype earlier
Product	Core
Plataform	x86 Linux
Type	Defect
Priority	Not set
Severity	Normal
Description	User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.1) Gecko/20020826
	Build Identifier: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.1) Gecko/20020826
	If the user earlier in the same browser-session has clicked cancel in a "click
	here to download plugin" dialog, Mozilla should not pop-up a new request for the
	same mimetype, atleast not within the same browser session.
	I guess this can be implemented by making
	modules/plugin/samples/default/unix/nullplugin.c contain a hash of all mimetypes
	that it has displayed dialogs for, and check for this early in makeWidget()
	Reproducible: Always
	Steps to Reproduce:
	1.Enter any page that contains a ref. to a mimetype that requires a plugin that
	is not installed
	2.click cancel on the download dialog
	3.re-enter the page
	Actual Results:
	The dialog reappares
	Expected Results:
	It should have remembered that I didn't want the plugin.

Figure 2.1: Bug report 175206 taken from Mozilla Bugzilla.



TC Number	155
TestDay	20161028
Generic Title	Downloads Dropmaker
Ctr Nr	4
Title	The downloads button works properly no matter where it's positioned in the browser UI
Steps	1. Launch Firefox with a clean profile. 2. Click menu [≡] ⇒ select Customize. 3. Drag the download button from the navigation bar and place it on the tabs bar. 4. Perform at least 3 downloads. 5. Open the Downloads Panel.
Expected Result	1. Firefox is successfully launched. 2. Customize mode is enable. 3. - The download button is placed on the tabs bar. - On Windows 7 (Aero), the button's color is white. - After the downloads are completed the downloads buttons turns blue. 4. The downloads button's color is changed.

Figure 2.2: Test case example taken from Mozilla Quality Assurance.

2.3 IR and ML Techniques

Information Retrieval (IR) can be defined as any approach that retrieves relevant textual *documents* in a corpus for a given *query*, and a document is considered relevant when it satisfies a need [8].

In the context of traceability between textual software artifacts, the use of IR is based on the premise that if engineers and developers refer to the same aspects of the system, a similar language is used across all artifacts [8]. In this research, bug reports are used as queries and test cases are used as documents.

2.3.1 Latent Semantic Indexing (LSI)

LSI [9] is a widely used algebraic IR technique based on the Vector Space Model (VSM) [8]. This technique vectorizes the corpus documents and queries by applying a selected weight scheme. One of the most common weight schemes used in LSI is the term frequency-inverse document frequency (*tf-idf*) [29], whose formula can be seen in Equation 2.1.

$$tf - idf(t, d, D) = tf(t, d) \times idf(t, D) \tag{2.1}$$

The first part of the formula (*tf(t,d)*) calculates the frequency of a term *t* in a given document *d*, so the more that term appears in the document, the higher the frequency value of the term (*tf*). The second part of the formula (*idf(t,D)*) calculates the amount of documents that the term *t* appears in the corpus of documents *D*, the rarer a terms is, the higher the inverse frequency value in the documents (*idf*), thus decreasing the weight of the term *t* within the entire corpus of *D*.

LSI is a technique that stands out for trying to reduce the noises present in the natural language by applying a mathematical dimensional reduction technique called Singular Value Decomposition (SVD) [9] in the matrix of term frequency. This reduction technique is needed to optimize the effectiveness of the LSI model, and make the search process process faster.

The similarity scores in that model are calculated by distance functions between the queries and target documents vectors. There are many distances functions, such as Dice coefficient [11] and Jaccard index [30], but in this study and in the baseline study the distance function used was the cosine similarity [11]. The author of the baseline work [12] does not explain the reasons for choosing the cosine distance function, despite mentioning in related works that other research varied the distance function. We also chose the cosine distance for our work so that the only differences between our research were the improvement techniques evaluated.

Figure 2.3 illustrates a concrete example of the LSI technique. Bug report and test cases are referred by their unique identifiers, the bug report 1267501 is referred as BR_1267501_SRC and the test cases as TC_13_TRG, TC_37_TRG, and TC_60_TRG. The "*SRC*" and "*TRG*" stands for *source* and *target*, respectively, to make explicit the direction of traceability recovery.

The process displayed in Figure 2.3 is performed after the preprocessing phase (tokenization, stemming, stop words removal, etc). The *Term-by-Document Matrix* (right of the figure) is created with the terms present in the test cases, the weights of the terms are calculated by the *tf-idf* scheme (Equation 2.1). The same process is performed in the bug report, creating the *Query Vector Subset* (left of the figure).

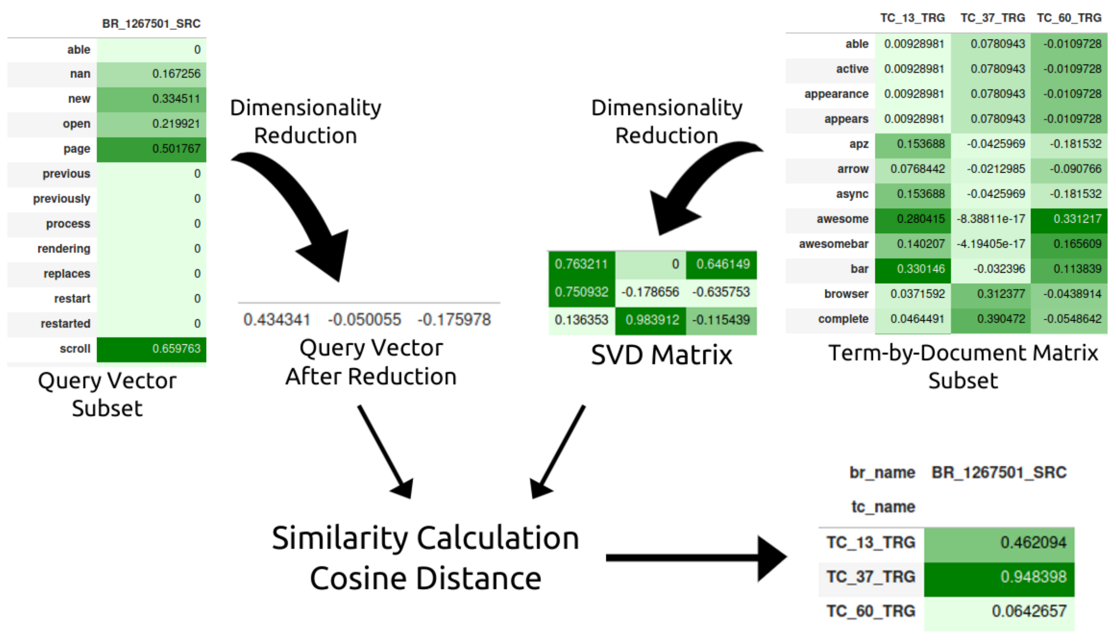
After the creation of the *Term-by-Document Matrix* and *Query Vector Subset* is applied the mathematical reduction technique SVD, as previously mentioned, generating the vector and matrix with a smaller amount of dimensions. After that, the last phase is the calculation of the similarity scores, using the cosine distance function, resulting in the similarity matrix between the bug report and the manual test cases.

2.3.2 Latent Dirichlet Allocation (LDA)

LDA is a generative statistical model [7], that technique models each collection of queries and documents as a set of topics, and each one of these topic is modeled as a set of probabilities. When the probabilities of a topic provides a representation of the documents or query, this topic is called a topic model.

From the topic model it is possible to estimate which topics are most representative for a given document, and assign a specific topic distribution for each document. And the similarity in this technique can be also calculated by the cosine similarity distance function.

The concrete example for the LDA technique is displayed in Figure 2.4. The *Term-by-*


Dimensionality Reduction
Dimensionality Reduction
Query Vector Subset
Query Vector After Reduction
SVD Matrix
Term-by-Document Matrix Subset
Similarity Calculation Cosine Distance

	BR_1267501_SRC
able	0
nan	0.167256
new	0.334511
open	0.219921
page	0.501767
previous	0
previously	0
process	0
rendering	0
replaces	0
restart	0
restarted	0
scroll	0.659763

0.434341	-0.050055	-0.175978

0.763211	0	0.646149
0.750932	-0.178656	-0.635753
0.136353	0.983912	-0.115439

	TC_13_TRG	TC_37_TRG	TC_60_TRG
able	0.00928981	0.0780943	-0.0109728
active	0.00928981	0.0780943	-0.0109728
appearance	0.00928981	0.0780943	-0.0109728
appears	0.00928981	0.0780943	-0.0109728
apz	0.153688	-0.0425969	-0.181532
arrow	0.0768442	-0.0212985	-0.090766
async	0.153688	-0.0425969	-0.181532
awesome	0.280415	-8.38811e-17	0.331217
awesomebar	0.140207	-4.19405e-17	0.165609
bar	0.330146	-0.032396	0.113839
browser	0.0371592	0.312377	-0.0438914
complete	0.0464491	0.390472	-0.0548642

br_name	BR_1267501_SRC
tc_name	
TC_13_TRG	0.462094
TC_37_TRG	0.948398
TC_60_TRG	0.0642657

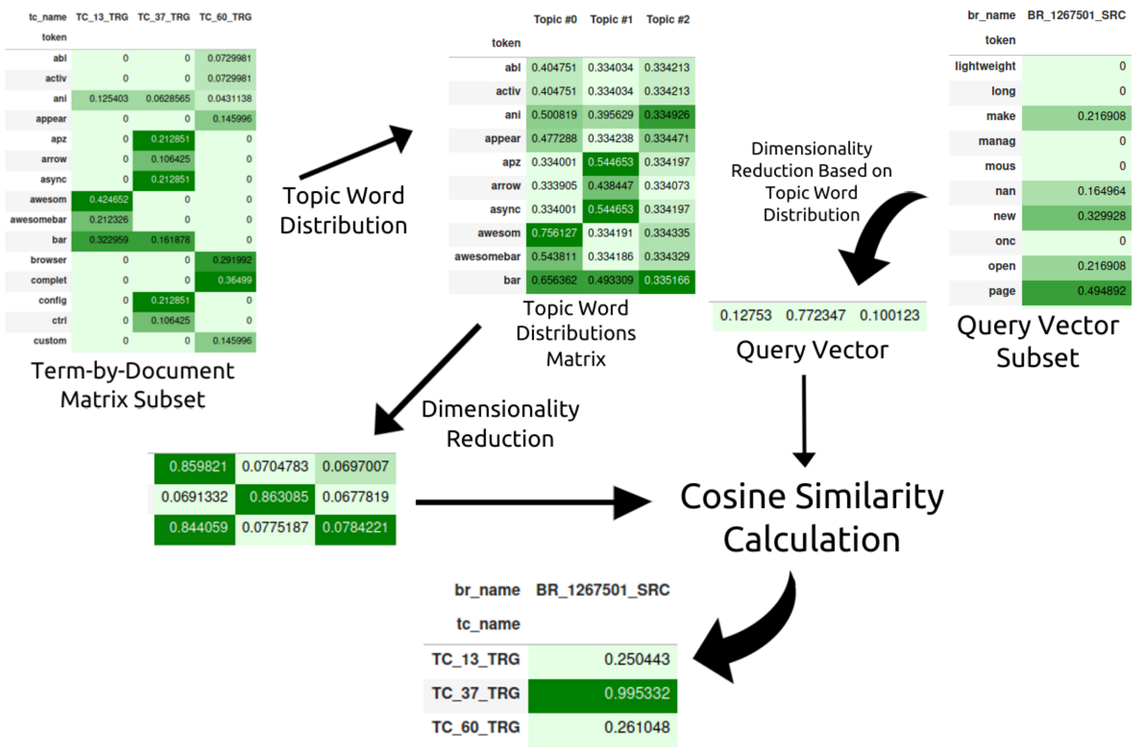
Figure 2.3: LSI example.

Document Matrix (left in the figure) is also calculated by the *tf-idf* scheme (Equation 2.1), but different from the LSI, this matrix is used to generate the *Topic Word Distributions Matrix*, containing the topic word distributions calculated by the LDA (three in this cases). The values contained in the *Topic Word Distributions Matrix* are probabilities. The *Query Vector* is generated by a similar process.

After the topic word calculation and creation of the *Topic Word Distributions Matrix* and *Query Vector*, a dimensional reduction operation is performed, resulting in the reduced version of both (the matrix and the vector). The last step is the calculation of the similarity scores, again using the cosine distance function.

2.3.3 Best Match 25 (BM25)

BM25 is a probabilistic model based on the weight function called Okapi-BM25 [31], described in the Equations 2.2 and 2.3. Probabilistic models are based on the *probabilistic ranking principle* [32], where documents in a collection must be ranked, in descending or-



Term-by-Document Matrix Subset

tc_name	TC_13_TRG	TC_37_TRG	TC_60_TRG
token			
abl	0	0	0.0729981
activ	0	0	0.0729981
ani	0.125403	0.0628565	0.0431138
appear	0	0	0.145996
apz	0	0.212851	0
arrow	0	0.106425	0
async	0	0.212851	0
awesom	0.424652	0	0
awesomebar	0.212326	0	0
bar	0.322959	0.161878	0
browser	0	0	0.291992
complet	0	0	0.36499
config	0	0.212851	0
ctrl	0	0.106425	0
custom	0	0	0.145996

Topic Word Distributions Matrix

	Topic #0	Topic #1	Topic #2
token			
abl	0.404751	0.334034	0.334213
activ	0.404751	0.334034	0.334213
ani	0.500819	0.395629	0.334926
appear	0.477288	0.334238	0.334471
apz	0.334001	0.544653	0.334197
arrow	0.333905	0.438447	0.334073
async	0.334001	0.544653	0.334197
awesom	0.756127	0.334191	0.334335
awesomebar	0.543811	0.334186	0.334329
bar	0.656362	0.493309	0.335166

Query Vector Subset

br_name	BR_1267501_SRC
token	
lightweight	0
long	0
make	0.216908
manag	0
mous	0
nan	0.164964
new	0.329928
onc	0
open	0.216908
page	0.494892

Query Vector

0.12753	0.772347	0.100123

0.859821	0.0704783	0.0697007
0.0691332	0.863085	0.0677819
0.844059	0.0775187	0.0784221

br_name	BR_1267501_SRC
tc_name	
TC_13_TRG	0.250443
TC_37_TRG	0.995332
TC_60_TRG	0.261048

Topic Word Distribution
Dimensionality Reduction Based on Topic Word Distribution
Dimensionality Reduction
Cosine Similarity Calculation

Figure 2.4: LDA example.

der, by the probability of their relevance in relation to a given query [35].

$$S_d(q) = \sum_{t \in q} W(t) \tag{2.2}$$

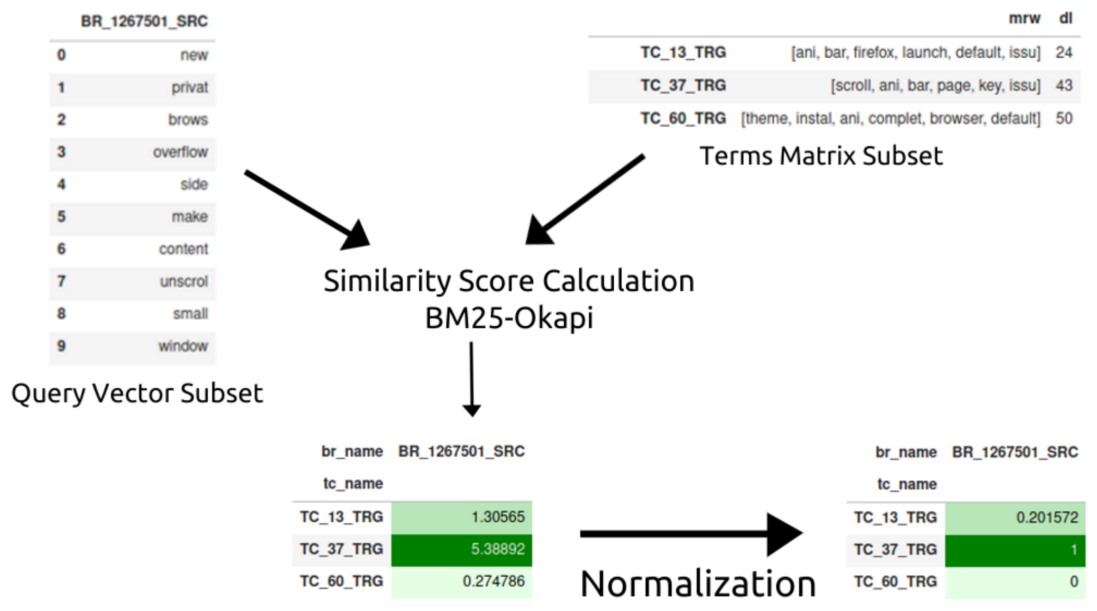
$$W(t) = \frac{TF_t(k_1 + 1)}{k_1((1 - b) + b \times \frac{DL}{AVGDL})} log(\frac{N}{ND_t}) \tag{2.3}$$

The Equation 2.2 calculates the final score of a document d in relation to a query q, which is the summation of the scores for each term t contained in the query q, in relation to the document d.

The individual score for each term t present in query q is calculated by the Equation 2.3, where TF_t is the frequency of the term t in the document d, *DL* is the document size, and *AVGDL* is the documents mean size. The variables k_1 and b are control parameters, k_1 controls the *term frequency* effect, and b controls the document size effect. In the second part of the Equation 2.3 N is the documents set size, and ND_t is the amount of documents in the set of documents that have the term t.

The Okapi-BM25 weight function can calculate similarities above the range [0, 1]. In order to compare the similarity scores calculated by the BM25 with the other techniques, the baseline work applies the normalization of the scores for the scale [0, 1]. The smallest similarity score becomes 0 and the highest becomes 1, the other values are calculated in relation to these two reference values, in the scale [0, 1].

Figure 2.5 presents the BM25 technique example. The *Query Vector Subset* and *Terms Matrix Subset* contain the terms present in the bug report and test cases, respectively, after the preprocessing phase (tokenization, stemming, etc). The similarity is calculated by the application of the Okapi-BM25 functions (Equations 2.2 and 2.3) for each pair of artifacts (bug report x manual test case). The last stage is the normalization of the results, as previously described.



	BR_1267501_SRC
0	new
1	privat
2	brows
3	overflow
4	side
5	make
6	content
7	unscrol
8	small
9	window

	mrw	dl
TC_13_TRG	[ani, bar, firefox, launch, default, issu]	24
TC_37_TRG	[scroll, ani, bar, page, key, issu]	43
TC_60_TRG	[theme, instal, ani, complet, browser, default]	50

br_name	BR_1267501_SRC
tc_name	
TC_13_TRG	1.30565
TC_37_TRG	5.38892
TC_60_TRG	0.274786

br_name	BR_1267501_SRC
tc_name	
TC_13_TRG	0.201572
TC_37_TRG	1
TC_60_TRG	0

Figure 2.5: BM25 example.

2.3.4 Word Vector

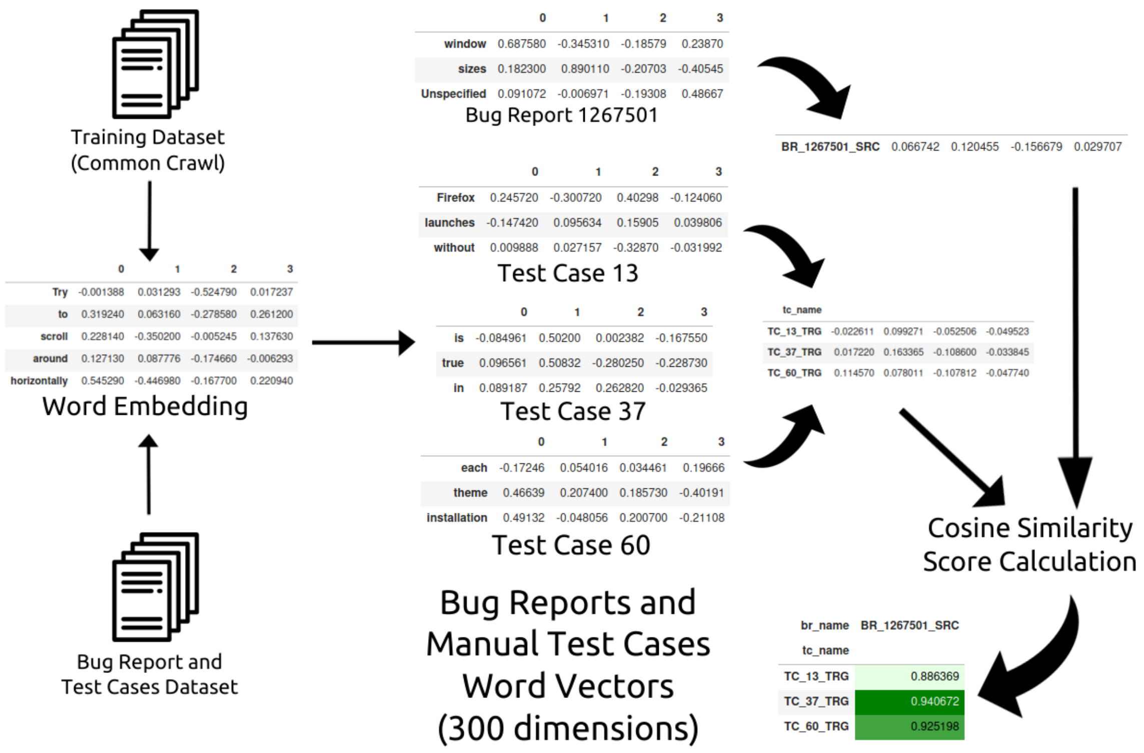
Word Vector (word embedding) is a general nomenclature attributed to a series of modeling and Deep Learning (DL) techniques in the area of Natural Language Processing (NLP) [16]. DL is a Machine Learning (ML) method based on Artificial Neural Networks [14]. These networks have a high number of hidden layers, some being very deep, and because of that they are able to capture many different patterns present in several types of dataset, such as images, texts, audio and videos, for example. The DL networks needs a large amount of data to be able to capture these patterns, and after being trained, they can be used in several activities, such as facial recognition in images and videos, textual translation, and speech recognition, among others.

The Word Vector techniques also vectorizes the set of received texts in a vector space. However, the vector representations computed by neural networks encode various language patterns and regularities [28]. This is the reason why these techniques have become popular for several applications such as text translation, speech recognition, and traceability recovery, because in addition to being able to estimate the syntactic similarity between texts, they also make it possible to capture the context of the words and their semantic similarities.

The algorithm applied on the baseline work was the Google's word2vec [27] open-source library, that receives as input a large amount of text to train the neural network and produce as output a vector space model, commonly with hundreds of dimensions. It is worth mentioning that word2vec is recognized as the state of the art in word embedding [13].

In the context of traceability recovery, the neural network can be retrained with the source and target artifacts datasets, so that the semantic and syntactic contexts present in the dataset are captured and the semantic relations updated. It can also be used to calculate the similarity between documents and queries in the same way as IR techniques, the algorithm produces vectors and vector space and similarity is calculated using the cosine function.

As in the previous techniques, we also illustrate a concrete example of the use of Word Vector, in Figure 2.6. A word embedding is shown on the left side of the figure, it has 300 dimensions and more than 1 million unique words, and it was trained with the Common Crawl Dataset (a textual corpus created based on texts from blogs, comments and news). The bug reports and test cases datasets are also added on the word embedding, without preprocessing. After that, the tokens of each document are grouped in matrices of word vectors, one matrix for each document. The next phase is the creation of reduced matrices, containing the calculation of the arithmetic mean of each grouped vectors, and each line represents a document. The last phase is the calculation of the cosine similarity for each pair of bug report and test cases.



	0	1	2	3
Try	-0.001388	0.031293	-0.524790	0.017237
to	0.319240	0.063160	-0.278580	0.261200
scroll	0.228140	-0.350200	-0.005245	0.137630
around	0.127130	0.087776	-0.174660	-0.006293
horizontally	0.545290	-0.446980	-0.167700	0.220940

	0	1	2	3
window	0.687580	-0.345310	-0.18579	0.23870
sizes	0.182300	0.890110	-0.20703	-0.40545
Unspecified	0.091072	-0.006971	-0.19308	0.48667

	0	1	2	3
Firefox	0.245720	-0.300720	0.40298	-0.124060
launches	-0.147420	0.095634	0.15905	0.039806
without	0.009888	0.027157	-0.32870	-0.031992

	0	1	2	3
is	-0.084961	0.50200	0.002382	-0.167550
true	0.096561	0.50832	-0.280250	-0.228730
in	0.089187	0.25792	0.262820	-0.029365

	0	1	2	3
each	-0.17246	0.054016	0.034461	0.19666
theme	0.46639	0.207400	0.185730	-0.40191
installation	0.49132	-0.048056	0.200700	-0.21108

BR_1267501_SRC	0.066742	0.120455	-0.156679	0.029707

tc_name				
TC_13_TRG	-0.022611	0.099271	-0.052506	-0.049523
TC_37_TRG	0.017220	0.163365	-0.108600	-0.033845
TC_60_TRG	0.114570	0.078011	-0.107812	-0.047740

br_name	BR_1267501_SRC
tc_name	
TC_13_TRG	0.886369
TC_37_TRG	0.940672
TC_60_TRG	0.925198

Figure 2.6: Word Vector example.

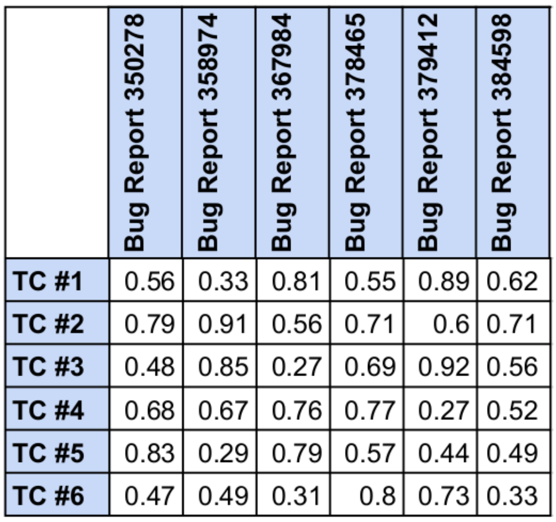
2.4 Matrices in the Traceability Context

In the context of traceability recovery, there are two types of matrices that are usually created during the process, namely: the similarity matrix and the traceability matrix. The similarity matrix is the output of IR techniques (or DL techniques applied in traceability context), this matrix contains the similarity scores. By applying a threshold value on these similarity scores is created the traceability matrix, that contains the traceability relations (trace or not trace) between the artifacts. A more detailed explanation follows.

2.4.1 Similarity Matrix

Similarity matrices are the output of traceability techniques (that uses similarity scores as a comparison parameter), these matrices hold in their cells the similarity score between pairs of software artifacts.

In Figure 2.7 we can see the representation of a generic similarity matrix that relates two sets of artifacts, bug reports and test cases. It is possible to see the similarity score for each pair of artifacts, and each similarity score present in the matrix is calculated by the traceability techniques, with each technique generating its own similarity matrix. For example, the similarity score between bug report 350278 and test case number 1 is 0.56, whereas the similarity score between bug report 350278 and test case number 2 is 0.79. These values are calculated by the traceability techniques (IR or DL), and with them it is possible to determine if there is a traceability link between these two artifacts, depending on the selected threshold value (it will be explained later).



	Bug Report 350278	Bug Report 358974	Bug Report 367984	Bug Report 378465	Bug Report 379412	Bug Report 384598
TC #1	0.56	0.33	0.81	0.55	0.89	0.62
TC #2	0.79	0.91	0.56	0.71	0.6	0.71
TC #3	0.48	0.85	0.27	0.69	0.92	0.56
TC #4	0.68	0.67	0.76	0.77	0.27	0.52
TC #5	0.83	0.29	0.79	0.57	0.44	0.49
TC #6	0.47	0.49	0.31	0.8	0.73	0.33

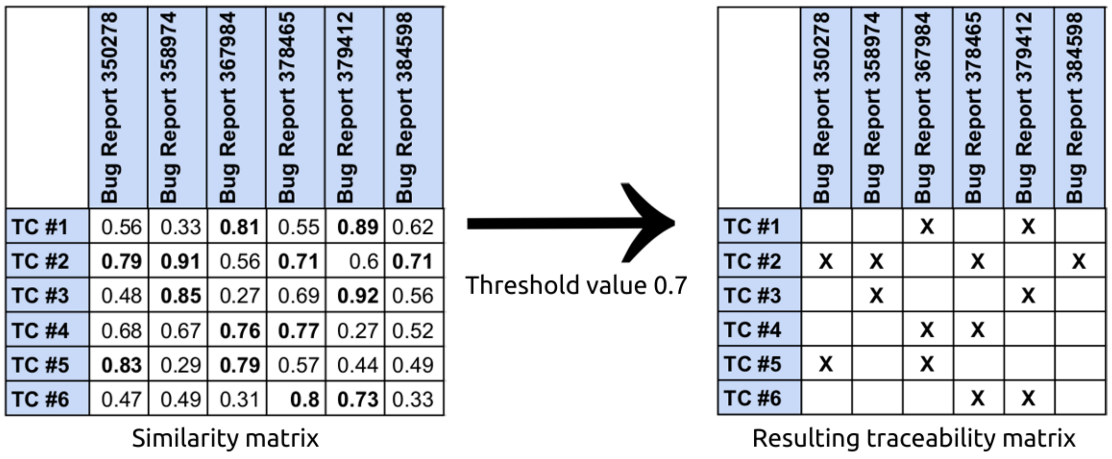
Figure 2.7: Example of a generic similarity matrix.

2.4.2 Traceability Matrix

In the context of software development, a traceability matrix is a document that aims to record the relationships between software artifacts. Usually in table format, it helps to determine the completeness of a relationship by correlating two artifacts using a many-to-many relationship comparison [15].

Traceability matrices are generated automatically when it comes to an automated traceability recovery process. These matrices are generated by applying a *threshold value* in a similarity matrix. The threshold value defines a minimum similarity score so that the relationship between a pair of artifacts is considered a traceability link or not.

In Figure 2.8 we can see the creation of a traceability matrix generated from the generic similarity matrix shown in Figure 2.7, by applying a threshold value equal to 0.7. Note that the values greater than or equal to the threshold value in the similarity matrix (in bold) have become traceability links between the pairs of artifacts (marked with **X**) in the generated traceability matrix. We use bug reports and test cases in this particular example to bring it closer to our research. However, similarity scores can be calculated among several other textual software artifacts.


Threshold value 0.7
Similarity matrix
Resulting traceability matrix

	Bug Report 350278	Bug Report 358974	Bug Report 367984	Bug Report 378465	Bug Report 379412	Bug Report 384598
TC #1	0.56	0.33	**0.81**	0.55	**0.89**	0.62
TC #2	**0.79**	**0.91**	0.56	**0.71**	0.6	**0.71**
TC #3	0.48	**0.85**	0.27	0.69	**0.92**	0.56
TC #4	0.68	0.67	**0.76**	**0.77**	0.27	0.52
TC #5	**0.83**	0.29	**0.79**	0.57	0.44	0.49
TC #6	0.47	0.49	0.31	**0.8**	**0.73**	0.33

	Bug Report 350278	Bug Report 358974	Bug Report 367984	Bug Report 378465	Bug Report 379412	Bug Report 384598
TC #1			X		X	
TC #2	X	X		X		X
TC #3		X			X	
TC #4			X	X		
TC #5	X		X			
TC #6				X	X	

Figure 2.8: Creation of a traceability matrix from a generic similarity matrix with a threshold value of 0.7.

2.5 Baseline Work

In this section we detail the technique [12] used as baseline study in our research. In order to better understand the approach in question, we discuss as follows its structure, the selected and used artifacts, and the evaluation method.

2.5.1 Definition

The baseline study, which we aim to complement, consists of an approach that performs three IR techniques (LSI, LDA and BM25) and a DL algorithm (Word Vector) in order to recover, study and evaluate the traceability between a set of bug reports and a set of test cases, using system features as intermediate artifacts.

In Figure 2.9 (taken from [12]) is shown the detailed scheme of the baseline approach. As we can see, the technique receives as input a set of bug reports, as source artifacts, and a set of test cases, as target artifacts, returning as output a set of traceability matrices for each traceability technique performed and each threshold and top value configuration.

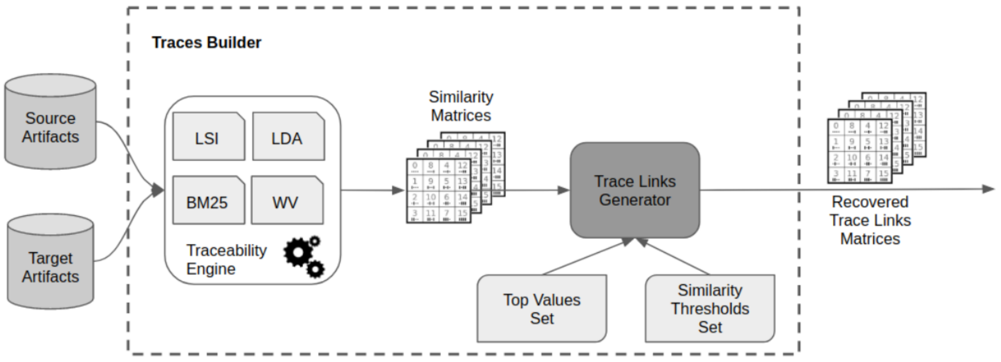


Figure 2.9: Baseline approach detailed scheme.

2.5.2 Artifacts

In the baseline study a total of 93 bug reports updated between 06/01/2016 and 12/31/2018 were selected from Mozilla Firefox's Bugzilla. The bug reports selection criteria were:

- Firefox version must be between 48 and 51;
- The bug reports status must be RESOLVED or VERIFIED;
- The bug report priority must be P1, P2, or P3 (the highest priority levels);
- The bug report resolution must be FIXED;
- The bug report severity must be major, normal, blocker, or critical.

These criteria were chosen to select the best quality bug reports from a continuous time frame, allowing to select a most relevant subset of bug reports in the entire dataset, following these criteria stated by the community. A "RESOLVED", "VERIFIED", and "FIXED" bug report already demonstrates a certain validation of the testing team. When selecting bug reports with higher priority, increase the chances of a bug report having a related manual test case. Also, these filters reduced the number of bug to be analyzed by the volunteers in the oracle creation study.

Feature Name	**Firefox Version**	**Number of TCs**
New Awesome Bar	48 and 50	13
Windows Child Mode	48	11
APZ - Async Scrolling	48	22
Browser Customization	49	6
PDF Viewer	49	8
Context Menu	49	31
Windows 10 Compatibility	49	6
Text to Speech on Desktop	49	2
Text to Speech in Reader Mode	49	8
WebGL Compatibility	49	3
Video and Canvas Renderization	49	2
Pointer Lock API	50	11
WebM EME Support for Widevine	50	6
Zoom Indicator	51	21
Downloads Dropmaker	51	18
WebGL2	51	3
FLAC Support	51	6
Indicator for Device Permissions	51	16
Flash Support	51	2

Table 2.1: Firefox Features from Baseline Study

The test cases were extracted from Firefox TestDays performed from 06/03/2016 to 01/06/2017. Altogether 195 test cases were collected from this period.

In total, 19 distinct Firefox system features were identified and tested during this period of TestDays. Each test case is related to a specific Firefox system feature, and this feature is explicitly indicated by the QA Team in the TestDays documents. in Table 2.1 is shown these system features.

2.5.3 Evaluation Method

To evaluate the traces recovered in the baseline study, it was necessary to manually create an oracle traceability matrix, that maps the bug reports to the Mozilla Firefox System features, and it was created from a crowdsourcing application with the help of volunteers and an "expert".

Oracle Matrix Creation

Altogether nine volunteers participated in the study for the creation of the oracle matrix, where all of them have Bachelor's Degrees in Computer Science. The aforementioned expert was the only one with previous knowledge of the Mozilla Firefox features, test cases and bug reports.

The creation process of the oracle matrix was made through an application that related, through the answers of the volunteers, the 93 selected bug reports with their respective features. The process was performed for the volunteers and the expert separately, thus creating two distinct matrices.

In order to have an agreement between the volunteers and the expert, the matrix chosen as the oracle for the evaluation of the techniques in the baseline study was the *Exp-Vol-Intersection*, the intersection between the volunteer's and expert's matrices.

Evaluation Metrics

To evaluate the baseline work a set of evaluation metrics were adopted.

Precision, *recall* and *F-Score* are common metrics used in traceability recovery [18]. Defined in the following Equations 2.4,2.5 and 2.6.

$$Precision = \frac{TP}{TP + FP} \tag{2.4}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.5}$$

$$F_\beta\text{-}Score = (1 + \beta^2) \times \frac{Precision \times Recall}{(\beta^2 \times Precision) + Precision} \tag{2.6}$$

TP is the number of True Positives, *FP* the number of False Negatives, and *FN* is the number of False Negatives in Equations 2.4 and 2.5.

In the Equation 2.6, F_β-$Score$ is the general version of the *F-Score metric*, in the baseline study the value assigned to β was 2 (two). The $F_2 - Score$ is a weighted version of the *F-Score*, which attributes the double of importance to the *Recall* score over the *Precision* [6]. The greater importance for recall is defined by the fact that it measures the completeness of the recovered traces, since a better recall means that more true positives traces have been recovered, and the approach is closer to recovering all traces.

Recovery Effort Index - REI

REI is a metric proposed by Antonion *et al.* [3] to evaluate the benefits of using an IR approach in a traceability recovery process, by estimating the percentage of the effort required to manually analyze the results achieved by the IR techniques, discarding the false positives, when comparing to the completely manual analysis (recovering the traces without any automation). The baseline study used a free adaptation of the original REI, calculated for each combination of Top Values and Similarity Thresholds, for each applied traceability technique, and comparing the resulting precision with the precision obtained by the oracle in the same scenario. The REI adaption formula used in baseline study can be seen in Equation 2.7.

$$REI_T = \frac{\sum_{i,j} \frac{OrcPrec}{T_{i,j}Prec}}{|S_{i,j}|} \tag{2.7}$$

Where REI_T is the REI of the technique *T*, the *OrcPrec* is the *Precision* obtained by the chosen oracle, $T_{i,j}Prec$ is the *Precision* of the technique *T* with the Top Value *i* and Similarity Threshold *j*, and $|S_{i,j}|$ is the cardinality of the set of combinations of Top Values and Similarity Thresholds.

The Goodness Levels

The Goodness levels were proposed by Hayes *et al.* [17] to establish boundaries for precision and recall metrics, classifying the level of traceability as *Acceptable*, *Good* or *Excellent*. The objective is to use this scale as a parameter to know if traceability methods are appropriate for tracing tasks. The baseline work used the reference values in the scale to additionally estimate the level of *Goodness* of the F_2-Score metric, in the same way made by Merten *et al.* [26].The values to each one of the levels can be seen in Table 2.2, and it is important to emphasize that values lower than those presented in the Table 2.2 are, in this scale, unacceptable.

	Acceptable	**Good**	**Excellent**
Precision	>20%	>30%	>50%
Recall	>60%	>70%	>80%
F_2-Score	>42.85%	>55.26%	>66.66%

Table 2.2: Goodness Levels [17].

ZeroR Predictor

A *Zeror Predictor/Classifiers* [20] was implemented in baseline work to serve as a baseline comparison. The *ZeroR Predictor* classifies a candidate trace between a pair of bug report and test cases as existent or not existent by simply predicting the majority class.

We do not use a ZeroR Prediction on our research because we already have a baseline work to compare.

2.5.4 Baseline Work General Result

Figure 2.10 displays the general results obtained for each applied traceability technique in baseline work [12]. The presented values of precision, recall and F_2-Score are the arithmetic mean for each combination of Top Values (top-10, top-20, and top-40) and similarity thresholds (from 0.0 to 0.9, increased by 0.1 in 0.1).

LSI obtained the best result among the traceability techniques achieving the values of 28.4% in precision, 48.6% in recall, and 35.2% in F_2-Score. However, even getting the best result among the traceability techniques, LSI failed to overcome the *ZeroR Predictor* in terms of recall (50.6%) and F_2-Score (40,1%), surpassing only the precision result. The other techniques had worse results than *ZeroR* in all the evaluated metrics (precision, recall, and F_2-Score). As previously mentioned, the *ZeroR* was used in the baseline work [12] as a baseline for comparing results.

Some LSI results can be highlighted, the technique reached more than 80% of recall in the top-40 in the thresholds values from 0.0 to 0.4. In the top-10, top-20, and top-40, with a threshold of 0.8, LSI achieved an precision greater than 40%. In addition to being the only technique to have reached the “Acceptable” level of the Goodness scale, in top-20 and top-40 with the threshold value 0.5.

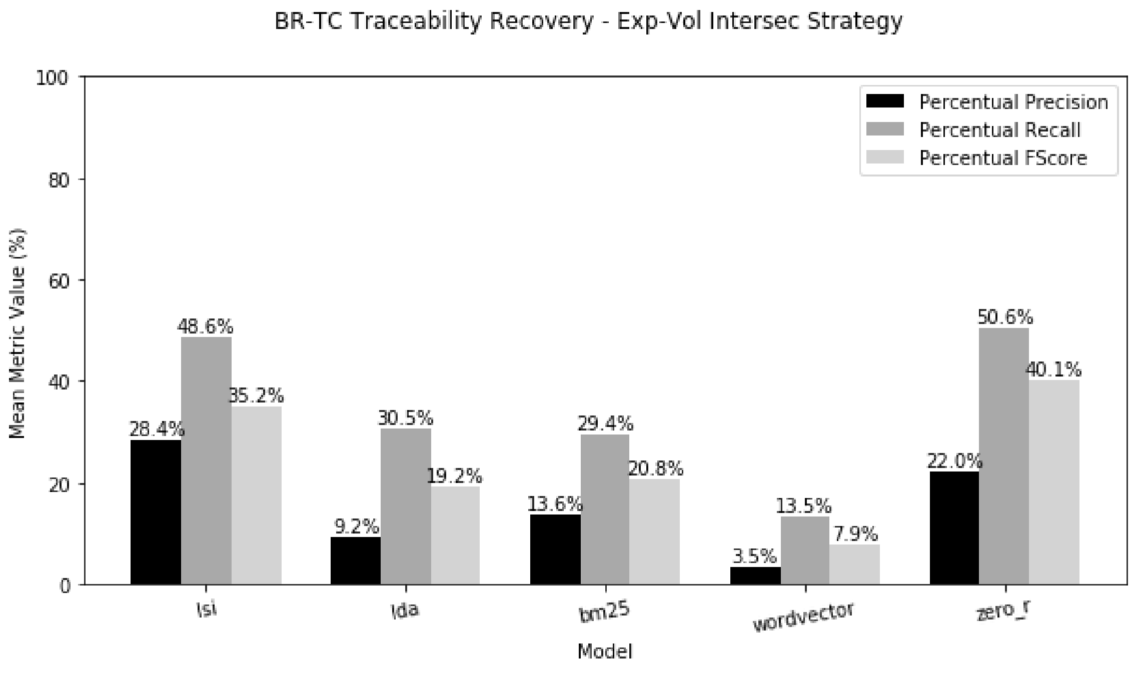


Figure 2.10: Baseline work general evaluation.

Chapter 3

Improvement Techniques

In order to extend the baseline work [12] we applied some preprocessing tasks in the bug reports dataset, as well as techniques that merge the traceability and similarity matrices generated by the recover techniques. These proposed techniques were developed in an attempt to improve the baseline work, it is worth mentioning that *we are not proposing a generic approach and we only tested on the baseline work* in question.

The techniques work in different parts of the baseline work. They can be divided into three types depending how and where they work.

1. Preprocessing approaches: information and text cleaning and spellchecker techniques;
2. Weighting approaches: bug report title and description duplication;
3. Hybrid approaches: merge of similarity and traceability matrices.

Figure 3.1 displays the scheme of the basic work with the areas where the improvement techniques act highlighted. The preprocessing and weighting approaches act in the bug reports text before they become inputs to the *Traces Builder* module. The hybrid approaches act in the similarity matrices generated as outputs from the *Traceability Engine* sub-module and in the traceability matrices generated as outputs from *Trace Links Generator* sub-module.

The improvement techniques are explained in detail as following.

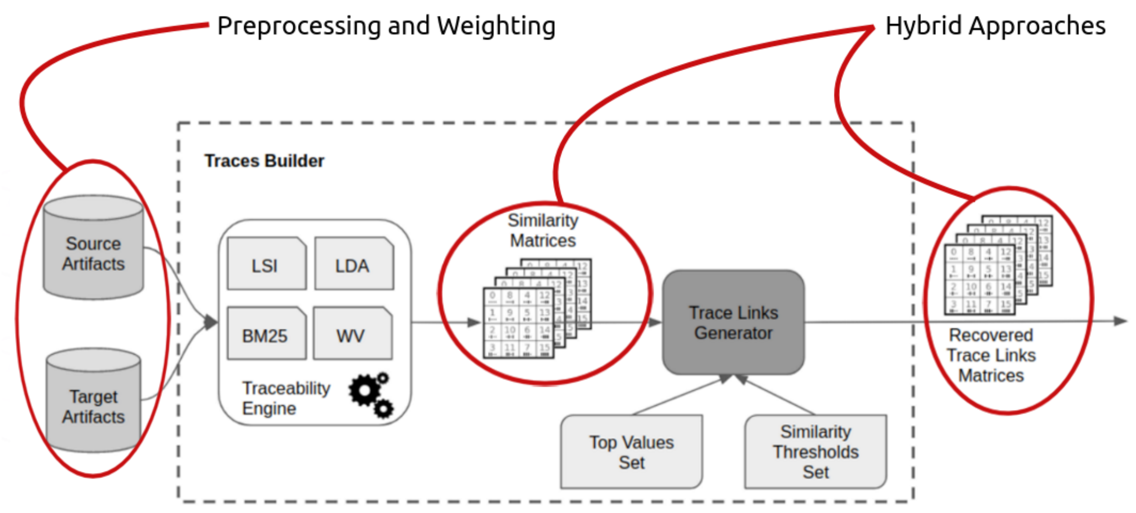


Figure 3.1: Acting areas and stages of the improvements techniques scheme.

3.1 Preprocessing Approaches

Textual preprocessing is an important step for Natural Language Processing (NLP) with IR and Machine Learning (ML) techniques [22]. The baseline work [12] applies some preprocessing techniques: tokenization, stop-word removal and stemming/lemmatization after the creation of the queries.

Despite this, there was no application of basic preprocessing techniques before the creation of the queries, such as punctuation removal and spelling correction.

3.1.1 Bug Report Text and Information Cleaning

The baseline work uses several information fields contained in the bug reports to create the queries used in the traceability recovery process. There are 17 information fields altogether: the bug report's ID, title, creation time, description and description's creation time, priority level, whiteboard, QA whiteboard, severity, resolution, status and confirmation status. In addition, some fields of information of the system related to the bug report are also used: the platform, component, version and product.

To find out whether all this information is either useful or noise-generating, we adopted in the query creation process only eight information fields.

Those fields were chosen manually, by analyzing the textual information contained in each of the fields. The criterion for choosing the fields is the presence of textual information relevant to the process of recovering traceability. The chosen fields were:

Bug report's ID is the numeric ID of a bug report, it is unique for each Bugzilla installation and identifies the bug report.

Title is the title created by the reporter, may contain relevant terms for the traceability process.

Bug report's description is the detailed (or not) description of the bug found by the bug report writer. Usually the field with the most textual information, may contain error messages, code snippets, execution steps, expected behavior and software's behavior.

Referring platform is a field related to the hardware where the bug was found. For instance, if a bug was found in a mobile device or a personal computer.

System component is the field that contain the component of the Mozilla Firefox where the bug was found by the reporter.

System version defines the version of the software the bug report was found. Test cases may contain the version of the software being tested.

Whiteboard is a free text line for adding tags, status information and keywords. It is a field with great potential to contain relevant textual information, however it is not always used.

QA whiteboard a field to adding tags, status information and keywords as the *whiteboard* field, but with restriction of the terms that can be added, the vocabulary is controlled by the QA team. It is also a field with great potential for textual relevance.

In order to further reduce the noise in the text, a textual cleaning was applied to remove punctuation marks in the text. This textual cleaning was performed in two steps:

1. Using the Python program language string library to separate words from punctuation marks;

2. Using the Python program language regex library to remove the punctuation marks.

To exemplify the text and information cleaning process, in Figure 3.2 we show two queries of the bug report 1267480 [1]. The first query was the one used in the baseline work, before the information and text cleaning process, the second one is the query used in this improvement technique, after applying the cleaning process.

Before	1267480 Intermittent browser/components/preferences/in-content/tests/ browser_security.jsThis test exceeded the timeout threshold. It should be rewritten or split up. Unspecified Preferences 48 Branch 2016-04-25T23: 58:35Z [stockwell fixed] nan https://treeherder.mozilla.org/logviewer.html #?job_id=26564567&repo=mozilla-inbound 2016-04-25T23:58:35Z RESOLVED Firefox P3 FIXED normal True BR_1267480_SRC
After	1267480 Intermittent browser components preferences in content tests browser_security jsThis test exceeded the timeout threshold It should be ewritten or split up Unspecified Preferences 48 Branch stockwell fixed nan https treeherder mozilla org logviewer html job_id 26564567 repo mozilla inbound

Figure 3.2: Bug report 1267480 query before and after applying the text and information technique.

Regarding the process information cleaning, we can notice at the end of the second query illustrated in Figure 3.2 the absence of some terms such as “P3”, “FIXED”, “normal”, “True”, and the repetition of the bug report identification, which were previously contained in the original query. As stated earlier, these fields have been removed because we believe that the textual information contained in them does not add to the information retrieval process, and may even be adversely affecting the result.

The textual cleaning process, as seen in Figure 3.2, removed several punctuation marks, which reduces the noise contained in the text. The process also separates some words previously joined by punctuation marks, such as “browser/components/preferences/in-content/browser_security.js”, and URLs.

[1]https://bugzilla.mozilla.org/show_bug.cgi?id=1267480

3.1.2 Spellchecker

Spellcheckers are programs created to check the spelling of a text, in order to find and correct typing errors. The idea of using a spellchecker on the bug report dataset assumes that if the bug reports are written by humans users, they may contain some spelling errors that can hinder the traceability recovery process.

The library used in the spell check process was the *pyspellchecker* [2], and it allows us to add words to its correct words set, and thus preventing these technical terms from being incorrectly replaced in the spell checker process.

To improve the spellchecker process (words merged by punctuation can also contain typos) we also performed the same punctuation removal process performed in the bug report text and information cleaning approach, described in Subsection 3.1.1.

3.2 Weight Approaches

The baseline work adopted the same weight to the terms contained in the different bug report information fields in the vector that represent each bug report. Merten *et al.* [26] applied IR techniques varying the weight for the terms present in different fields of the bug reports, achieving a slight result improvement in the techniques. In order to find out if terms contained in the different fields (in this case the title and the description) of the Mozilla Firefox bug reports have the same impact in the traceability process, queries with duplicated titles and queries with duplicated descriptions were created and evaluated. The purpose of duplication is to increase the frequency of terms in the query text. These two fields were chosen because they are the ones that most present textual information.

We also tested to increase the number of repetitions, but when we tried to triple the fields the results obtained were very negative, so we discard the increase in repetitions.

To illustrate the changes made by this improvement technique, in Figure 3.3 we have the original query for the base work of bug report 1181835, in it we have the title of the bug report in green and its description in blue. In Figure 3.4 it is displayed the query of the same bug report, but after the title duplication, in green. And in Figure 3.5 is shown the same bug report query after the description duplication, in blue.

[2]https://pypi.org/project/pyspellchecker/

1181835 Provide a UI for migrating users' add-ons to webextensions Unspecified Extension Compatibility 49 Branch 2015-07-08T23:01:45Z [UX] triaged nan We're still not exactly sure how this would work, but I think the basic idea is as follows. On startup, you would be shown a tab with a list of your add-ons. For each add-on, we would show its status: - e10s compatible (doesn't use CPOWs at all) - e10s functional (works, uses CPOWs, probably will make Firefox slower) - broken with e10s For each add-on not in the first category, we'd like to provide alternatives: - disable the add-on - upgrade to a beta version that works better - upgrade to a slightly different add-on (Firebug 2 ->Firebug 3 for example) We would probably also show this UI whenever the status of an add-on changes (for example a disabled add-on being made e10s-compatible). 2015-07-08T23:01:45Z RESOLVED Firefox P2 FIXED normal True BR_1181835_SRC

Figure 3.3: Query of the bug report 1181835 in baseline work.

1181835 Provide a UI for migrating users' add-ons to webextensions Provide a UI for migrating users' add-ons to webextensions Unspecified Extension Compatibility 49 Branch 2015-07-08T23:01:45Z [UX] triaged nan We're still not exactly sure how this would work, but I think the basic idea is as follows. On startup, you would be shown a tab with a list of your add-ons. For each add-on, we would show its status: - e10s compatible (doesn't use CPOWs at all) - e10s functional (works, uses CPOWs, probably will make Firefox slower) - broken with e10s For each add-on not in the first category, we'd like to provide alternatives: - disable the add-on - upgrade to a beta version that works better - upgrade to a slightly different add-on (Firebug 2 ->Firebug 3 for example) We would probably also show this UI whenever the status of an add-on changes (for example a disabled add-on being made e10s-compatible). 2015-07-08T23:01:45Z RESOLVED Firefox P2 FIXED normal True BR_1181835_SRC

Figure 3.4: Query of the bug report 1181835 with title duplication.

1181835 Provide a UI for migrating users' add-ons to webextensions Unspecified Extension Compatibility 49 Branch 2015-07-08T23:01:45Z [UX] triaged nan We're still not exactly sure how this would work, but I think the basic idea is as follows. On startup, you would be shown a tab with a list of your add-ons. For each add-on, we would show its status: - e10s compatible (doesn't use CPOWs at all) - e10s functional (works, uses CPOWs, probably will make Firefox slower) - broken with e10s For each add-on not in the first category, we'd like to provide alternatives: - disable the add-on - upgrade to a beta version that works better - upgrade to a slightly different add-on (Firebug 2 ->Firebug 3 for example) We would probably also show this UI whenever the status of an add-on changes (for example a disabled add-on being made e10s-compatible). We're still not exactly sure how this would work, but I think the basic idea is as follows. On startup, you would be shown a tab with a list of your add-ons. For each add-on, we would show its status: - e10s compatible (doesn't use CPOWs at all) - e10s functional (works, uses CPOWs, probably will make Firefox slower) - broken with e10s For each add-on not in the first category, we'd like to provide alternatives: - disable the add-on - upgrade to a beta version that works better - upgrade to a slightly different add-on (Firebug 2 ->Firebug 3 for example) We would probably also show this UI whenever the status of an add-on changes (for example a disabled add-on being made e10s-compatible). 2015-07-08T23:01:45Z RESOLVED Firefox P2 FIXED normal True BR_1181835_SRC

Figure 3.5: Query of the bug report 1181835 with description duplication.

3.3 Hybrid Approaches

One of the observed results in the baseline work [12] was that the four techniques applied in the process of traceability recovering had distinct behaviors in relation to the recovered traces. It was observed that the sets of recovered traces by each technique contained several traces that were disjoint in relation to the other techniques.

In others words, there were several true positives traces that were recovered by only one of the four applied traceability techniques. Motivated by that, we decided to investigate the combination of the four traceability technique by merging their similarity and traceability matrices in two hybrid improvement techniques, detailed in sequel.

3.3.1 The Similarity Matrices Merge

The first hybrid approach applied was the weighted merge of the traceability matrices. A similarity matrix was thus produced from the weighted arithmetic mean of the similarity matrices by the four traceability techniques - each element in the hybrid similarity matrix holds the value of the weighted arithmetic mean of the elements in each technique similarity matrix.

The weight for each technique was defined by the proportion of its F_2-Score from the general results of the baseline study (Figure 2.10), so that the techniques that performed better have a greater weight in their similarity scores. Weights were calculated as follows: the sum of the F_2-Score results of the traceability techniques was defined as 1 (100%), from this value the proportion of the results of each traceability techniques were calculated in relation to that value. For example, the weight of the LSI is 0.425, this means that the percentage proportion obtained by the technique was 42.5%, whereas the word vector was 9.5%. The general result gives us a view of how each of the techniques behaves in all scenarios, since it is an arithmetic mean of the results of all the cuts and threshold values configurations. However, this is the least generic technique and the one that has the most risk of causing overfitting, since we use data from the result of an approach (baseline work [12]) to try to improve the same approach. The weights assigned to each technique can be seen in Table 3.1. The LSI technique achieved a better result than the others traceability techniques in the baseline work, in all scenarios, so it gets the biggest weight (0.425), unlike word

Table 3.1: Weights assigned for each traceability technique.

LSI	0.425
LDA	0.23
BM25	0.25
Word Vector	0.095

vector, which obtained the worst result and consequently the lowest weight (0.095). LDA and BM25 obtained average weights, 0.23 and 0.25, respectively.

The final similarity score for each pair of artifacts is calculated as show in Equation 3.1:

$$S_F = (S_{LSI} \times 0.425) + (S_{LDA} \times 0.23) + (S_{BM25} \times 0.25) + (S_{WV} \times 0.095) \qquad (3.1)$$

Where S_F is the final similarity value, S_{LSI} is the LSI original similarity value, as with the similarity score of the other traceability techniques in S_{LDA} (for LDA), S_{BM25} (for BM25), and S_{WV} (for Word Vector) The weights present in Equation 3.1 are the same weights displayed in Table 3.1, and the similarity values for each traceability technique are taken from the respective similarity matrix of the technique.

Thus, the technique with the best results (the LSI technique) may have a positive influence on the similarity scores of the techniques that obtained lower performances. However, the opposite can occur and techniques with lower performance can negatively influence the similarity values obtained by the best technique.

3.3.2 The Traceability Matrices Merge

The traceability matrices merge technique analyzes the baseline work as a single technique, instead of analyzing each of the four traceability techniques independently.

Three traces selection strategies, with different levels of sensitivity, were evaluated, each one resulting a new traceability matrix containing the traces recovered by:

1. At least one of the four traceability techniques;
2. At least two of the four traceability techniques;

3. At least three of the four traceability techniques.

We only evaluated those three strategies, and we did not use a fourth strategy selecting only the traces recovered by the four traceability technique because it differs from what traceability matrices merging technique proposes, which is precisely trying to take advantage of the four techniques as one, even with the differences in the selection of traces that each technique presents.

These strategies were chosen so that it was possible to evaluate the precision and recall values obtained in distinct criteria scenario, since false positives and false negatives can also be inserted in the merged matrices depending on the level of selectability of each strategy.

Chapter 4

Research Method

This chapter presents the method applied in this research, starting with the exposure of the hypotheses from the improvement techniques proposed by this work, followed by the research questions used to evaluate the work are also defined. Finally, the experimental procedure session is decribed, where we address the artifacts used and the evaluation metrics adopted by our work.

4.1 The Improvement Techniques Hypotheses

Each of the proposed improvement techniques previously mentioned were based on hypotheses elaborated by us through the analysis of the study and results of the baseline work.

During the discussion of the results of the improvement techniques, these hypotheses are tested. The hypotheses are as follows.

Hypothesis 1: an excessive textual information of a bug report contained in queries can impair the information retrieval process. By reducing this noise, we can achieve a better result.

Hypothesis 2: bug reports are written by humans, so it is possible for typos to occur, which can impair the traceability recovery process.

Hypothesis 3: some fields in the bug report may be more relevant to the traceability recovery process, so if we increase the weight of these fields in the creation of queries we can improve the results.

Hypothesis 4: if the traceability techniques used in the baseline work show different behaviors when recovering the traces, it is possible to join their effort.

The *Hypothesis 1* was responsible for the proposal of the "*Bug Report Text and Information Cleaning*" technique (in 3.1.1); the *Hypothesis 2* for the the proposal of the "*Spellchecker*" technique (in 3.1.2); the *Hypothesis 3*, in its turn, was responsible for the the proposal of the "*Title Duplication*" and "*Description Duplication*" techniques (in 3.2); and for last, the *Hypothesis 4* was responsible for the the proposal of the two hybrid approaches, the "*Similarity Matrices Combination*" and "*Traceability Matrices Merge* techniques (in 3.3).

4.2 Research Questions

In order to assess the efficiency of the proposed improvement techniques and test the hypotheses, we performed an experimental study that discuss the answers to the following research questions:

RQ1: *What is the effect of applying the text and information cleaning technique on the baseline work?* Following the Hypothesis 1, the cleaning procedure is expected to improve the result obtained, once the not relevant text is removed from queries used.

RQ2: *What is the effect of using an English spellchecker on the baseline work?* Following the Hypothesis 2, a spellchecker can correct human mistakes in typing bug reports; this would help traceability techniques point out trace links otherwise missed due to this kind of issue.

RQ3: *What is the effect of doubling the bug report title or description on the baseline work?* Following the Hypothesis 3, we explore different weights for fields that compose the analyzed artifacts, applying higher weights to the title and description fields. Our hypothesis is that they have higher relevance, and their terms should have a more significant influence in the traceability recovery process.

RQ4: *What is the effect of combining the three techniques with positive results (textual cleaning, spellchecker and doubling the title)?* We suppose the combined application of these procedures may improve the results obtained with the baseline approach. The techniques were selected to be combined because they achieved positive results, and it is the only combination of improvement techniques evaluated in this research.

RQ5: *What are the results obtained by the hybrid techniques?* The baseline work study showed that the set of traces recovered by each traceability techniques were *disjoint*, following the Hypothesis 4, we combined the effort of all traceability techniques to check the benefits and drawbacks of all techniques applied.

4.3 Experimental Procedure

4.3.1 Artifacts Dataset

As an open-source project, Mozilla Firefox artifacts can be found freely on the internet. The dataset used in the baseline work [12] was made publicly available by the author. Part of our evaluation is made by comparing the results of the baseline work with those obtained after applying the improvement techniques, to ensure that the only difference between the experiments is the application of the improvement techniques, we chose to use the same dataset of bug reports and manual test cases.

The test cases were extracted from Firefox TestDays carried out from 06/03/2016 to 01/06/2017. Altogether 195 test cases were collected from this period. These test cases are made available by the Mozilla Quality Assurance team[1].

A total of 93 bug reports updated between 06/01/2016 and 12/31/2018 were selected from the Mozilla's Bugzilla[2]. The bug report selection criteria, defined and used by the baseline work, were:

- Firefox version must be between 48 to 51;
- Status must be RESOLVED or VERIFIED;

[1]https://quality.mozilla.org/
[2]https://bugzilla.mozilla.org/

- The priority must be P1, P2 or P3 (the highest priority levels);
- Resolution must be FIXED.

These criteria were chosen to select the best quality bug reports from a continuous time frame, and that was related to the test cases from the available Firefox documentation. Also, the baseline work prioritized bugs already fixed (or verified) - the development team was able to understand and fix/check the bug report - and classified as a high priority bug for the Mozilla development team, so it was possible to improve the chances to have access to the most relevant and well-described bug reports.

4.3.2 Implementation Tools

The traceability recover process is is divided into two phases, where different open-source code libraries are used. The first phase, preprocessing, it is used *Python's NLTK*[3] (Natural Language Toolkit), a platform for natural language processing (NLP). This library performs the process of tokenization, stop-word removal and stemming/lemmatization.

The second phase is the application of the traceability techniques. For LSI and LDA are used the implementation of these techniques from the *Scikit-Learn Data Analysis Toolkit*[4] library; for BM25 we used the *Gensim Library*[5]; and for Word Vector, it is used *SpaCy Library*[6].

The source codes for the implementation of traceability techniques were also made available by the author of the baseline work and reused in this research, so that there were no differences in the implementation of recovery techniques.

4.3.3 Research Evaluation

The baseline work evaluates its approach in some threshold values (the similarity limit value for a relationship between artifacts to be considered a trace or not): from 0.0 to 0.9, but the detailed evaluation is performed in two scenarios: (i) with the threshold value 0.0; and

[3]https://www.nltk.org
[4]https://scikit-learn.org/stable/
[5]https://radimrehurek.com/gensim/
[6]https://spacy.io/

(ii) with threshold value 0.9. The value 0.0 favors a higher amount of links to be recovered (higher recall), while the 0.9 value favors their precision.

In the two scenarios mentioned above three top values were chosen by the baseline study: Tops 10, 20, and 40. The baseline work observed that the average number of test cases linked to a bug report in this dataset is no larger than 40 [12], so all related bug report supposedly could be recovered in Top-40 by any of the evaluated techniques.

As our goal is to evaluate the proposed improvement techniques by comparing with the results obtained by the baseline work, we evaluated our improvement techniques in the same two threshold scenarios, with the same top values applied in baseline work.

With the exception of the test where we evaluated three combined improvement techniques, each one of the proposed techniques in this research is performed independently and separately.

4.4 Evaluation Metrics

In order to proper compare the results obtained after application of the improvement techniques, we adopted the same metrics as the baseline work (precision, recall, F_2-Score and goodness scale), previously mentioned in Subsection 2.5.3.

Chapter 5

Results and Discussion

In this chapter, we expose the results obtained by each of the improvement techniques and compare them, by research questions, to the results obtained in the baseline work. In addition to discussing the results obtained and the hypotheses previously raised.

The evaluation is made in the two scenarios , the first scenario with a 0.0 threshold value and the second scenario with a 0.9 threshold. We perform the detailed evaluation on those two scenarios because those scenarios are also explored in baseline work, and this allow us to make a better comparison of the results. The evaluation is made as follows: (i) a comparison of the general result of the traceability techniques (LSI, LDA, BM25, and Word Vector) before and after the application of the improvement techniques; (ii) an analysis of changes in the set of true positives traces that were missed by all traceability techniques (false negatives); and (iii) an analysis of changes in the set of true positives traces that were recovered by all traceability techniques.

5.1 RQ1 - What is the effect of applying the text and information cleaning technique on the baseline work?

As can be seen in Figure 5.1, by applying the text and information cleaning technique it was achieved a slight improvement in all four traceability techniques comparing with the baseline work results, both in true positives and false negatives, since there was an increase in precision and recall rates. As a reminder, the general result displayed in the graphs is an arithmetic average between the results obtained in all cuts (top-10, top-20 and top-40) with all threshold settings (0.0 to 0.9, adding 0.1 in 0.1). It is not a *top-k* result.

The LSI traceability technique had a 1.4% increase in the precision rate, 2.4% increase in recall, and a 2.1% increase in the F_2-Score rate; LDA had an increase of 0.6% in precision, 1.7% in recall, and 1.2% in F_2-Score; the BM25 traceability technique had the precision increased in 0.3%, the recall in 0.2%, and the F_2-Score in 0.4%; finally, Word Vector had a increase of 1.4% in precision, 4.7% in recall, and 3% in F_2-Score.

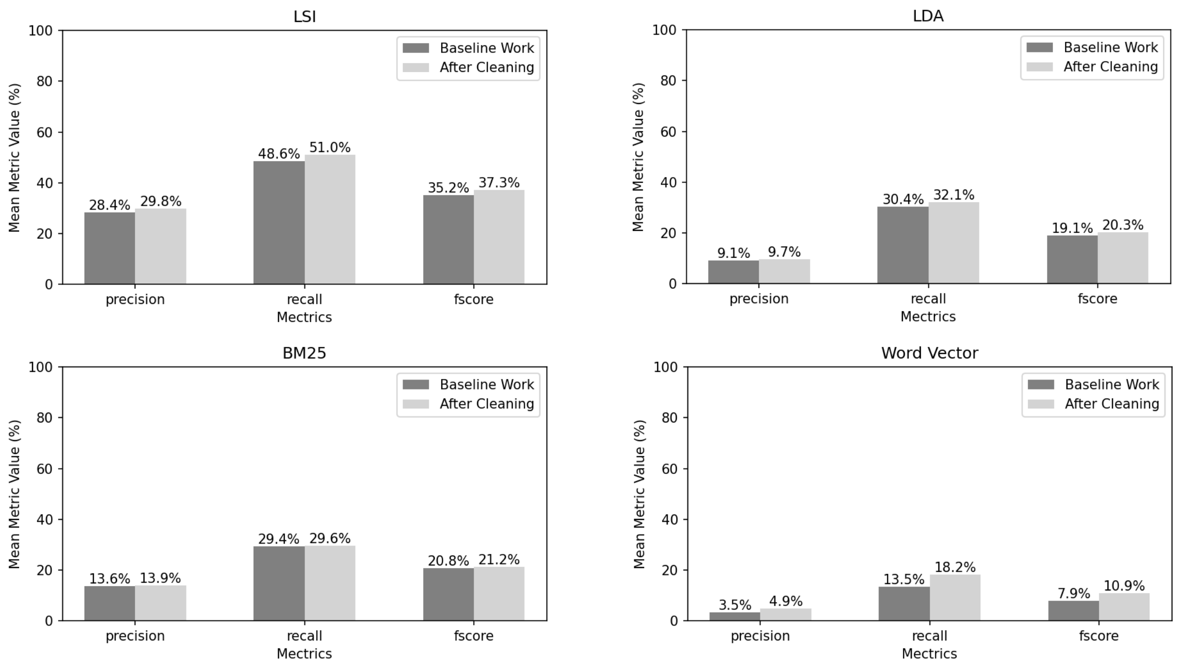


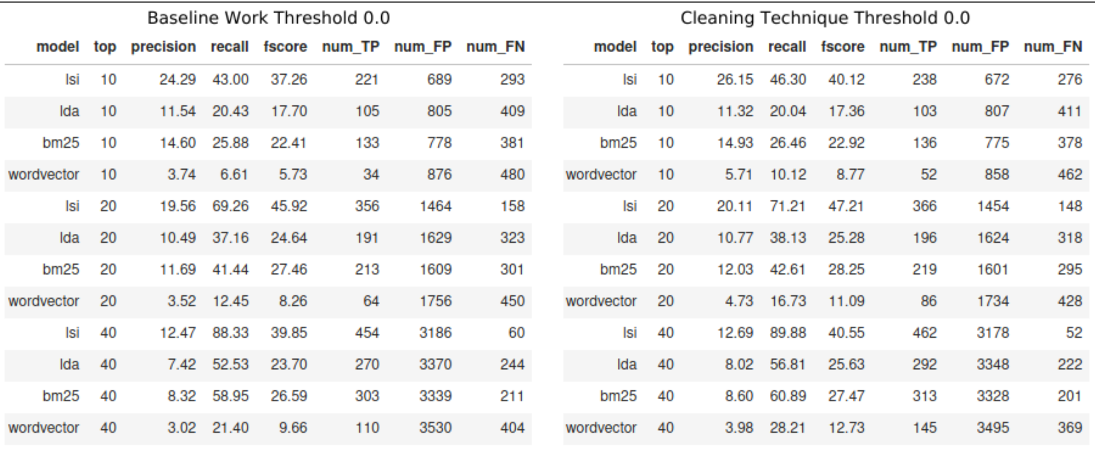
Figure 5.1: Baseline work and cleaning technique general results.

5.1.1 First Scenario - Similarity Threshold 0.0

First Scenario General Result

Figure 5.2 displays the results of baseline work traceability techniques alongside the results obtained after applying the improvement text and information cleaning technique, in the first scenario and all top cuts.

With the exception of the LDA in the top-10, there is an improvement in results in all techniques and cuts. One can see an increase in the rates of precision, recall, F_2-Score, and number of true positives trace, and decrease in the number of false positives and negatives.



Baseline Work Threshold 0.0

model	top	precision	recall	fscore	num_TP	num_FP	num_FN
lsi	10	24.29	43.00	37.26	221	689	293
lda	10	11.54	20.43	17.70	105	805	409
bm25	10	14.60	25.88	22.41	133	778	381
wordvector	10	3.74	6.61	5.73	34	876	480
lsi	20	19.56	69.26	45.92	356	1464	158
lda	20	10.49	37.16	24.64	191	1629	323
bm25	20	11.69	41.44	27.46	213	1609	301
wordvector	20	3.52	12.45	8.26	64	1756	450
lsi	40	12.47	88.33	39.85	454	3186	60
lda	40	7.42	52.53	23.70	270	3370	244
bm25	40	8.32	58.95	26.59	303	3339	211
wordvector	40	3.02	21.40	9.66	110	3530	404

Cleaning Technique Threshold 0.0

model	top	precision	recall	fscore	num_TP	num_FP	num_FN
lsi	10	26.15	46.30	40.12	238	672	276
lda	10	11.32	20.04	17.36	103	807	411
bm25	10	14.93	26.46	22.92	136	775	378
wordvector	10	5.71	10.12	8.77	52	858	462
lsi	20	20.11	71.21	47.21	366	1454	148
lda	20	10.77	38.13	25.28	196	1624	318
bm25	20	12.03	42.61	28.25	219	1601	295
wordvector	20	4.73	16.73	11.09	86	1734	428
lsi	40	12.69	89.88	40.55	462	3178	52
lda	40	8.02	56.81	25.63	292	3348	222
bm25	40	8.60	60.89	27.47	313	3328	201
wordvector	40	3.98	28.21	12.73	145	3495	369

Figure 5.2: Baseline work and text and information cleaning results: Threshold 0.0 scenario.

	Traces Missed by All Techniques	
	Baseline	Cleaning
Top 10	214	198
Top 20	104	87
Top 40	33	22

Table 5.1: Amount of traces missed by all techniques - Baseline work and Cleaning.

The changes are further detailed as follows.

Traces Missed by All Techniques

The cleaning technique achieved a decrease in the amount of true positives traces that were *missed by all the four traceability techniques* (false negatives) in all cuts. In Table 5.1 it is possible to see the numbers of missed traces of the baseline work and the cleaning technique for all three cuts. Compared with the baseline study results, the amount decreased from 214 to 198 traces in the top-10 cut, 104 from to 87 in top-20 cut, and from 33 to 22 in the top-40 cut.

In the baseline study, those missed traces are related to six bug reports and three system features (*New Awesome Bar*, *Browser Customization*, and *Context Menu*) in the top-40 cut (the largest cut).

After the cleaning process, only five bug reports were related to these missed traces, and the number of system features dropped to two (*New Awesome Bar* and *Browser Cus-*

tomization). With the cleaning technique, we recover all the traces of bugs reports 1293308 and 1299458 (related to the *New Awesome Bar* and *Context Menu features*, respectively), differently from the baseline study.

Bug report 1299458 ("Telemetry data from Search bar is not properly collected when searching in new tab from context menu") had two true positives traces recovered that were missed by all traceability techniques in baseline work - the only bug report related to the Context Menu feature. It happens because that bug contains in its description some relevant words merged by hyphen and periods. In Figure 5.3 we can notice, for instance, the words "context" and "searchbar" merged by hyphens with other terms in "{"other.oneoff-context-searchbar":{"unknown":{"tab-background":1}}".

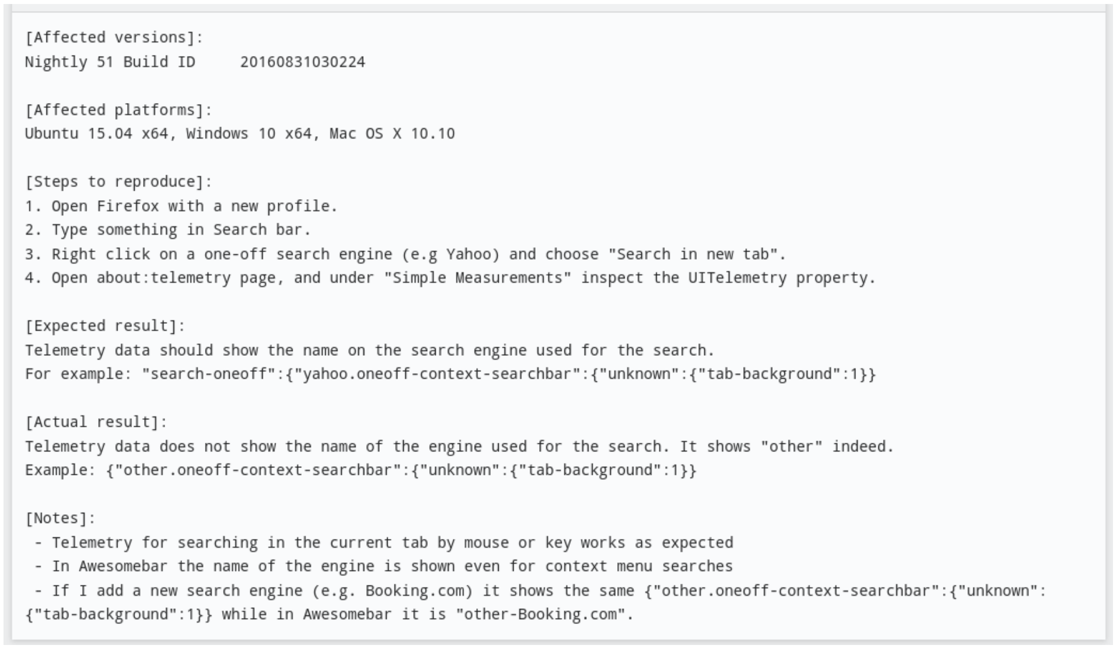

```
[Affected versions]:
Nightly 51 Build ID     20160831030224

[Affected platforms]:
Ubuntu 15.04 x64, Windows 10 x64, Mac OS X 10.10

[Steps to reproduce]:
1. Open Firefox with a new profile.
2. Type something in Search bar.
3. Right click on a one-off search engine (e.g Yahoo) and choose "Search in new tab".
4. Open about:telemetry page, and under "Simple Measurements" inspect the UITelemetry property.

[Expected result]:
Telemetry data should show the name on the search engine used for the search.
For example: "search-oneoff":{"yahoo.oneoff-context-searchbar":{"unknown":{"tab-background":1}}

[Actual result]:
Telemetry data does not show the name of the engine used for the search. It shows "other" indeed.
Example: {"other.oneoff-context-searchbar":{"unknown":{"tab-background":1}}

[Notes]:
 - Telemetry for searching in the current tab by mouse or key works as expected
 - In Awesomebar the name of the engine is shown even for context menu searches
 - If I add a new search engine (e.g. Booking.com) it shows the same {"other.oneoff-context-searchbar":{"unknown":
{"tab-background":1}} while in Awesomebar it is "other-Booking.com".
```

Figure 5.3: Bug report 1299458 description.

The same happens in bug report 1293308 ("System font setting respected by location bar but ignored by location bar's autocomplete drop-down list"), as in "Urlbar popup/drop-down font size is small." (excerpt taken from the bug report description), where the "drop-down" word merges with the word "popup". In these two cases, the punctuation removal process separates important terms from other words, allowing to represent them more appropriately in the vectors, increasing hence their relevance in the trace recovery process.

However, bug report 1279864, related to the *New Awesome Bar feature*, had three true positives traces missed by all traceability techniques. It does not have relevant terms joined

	Traces Recovered by All Techniques	
	Baseline	Cleaning
Top 10	7	14
Top 20	24	35
Top 40	65	86

Table 5.2: Amount of traces recovered by all techniques - Baseline work and Cleaning.

by punctuation marks in its description. In fact, the textual cleaning process did not affect the trace recovery process in this case. The change was due to the removal of the information fields present in the information and text cleaning technique, as described in Section 3.1.1.

Traces Recovered by All Techniques

Considering the number of true positives traces recovered by all techniques, there was an increase in all applied cuts when comparing with the result obtained in the baseline study, as we can see in Table 5.2. These increases are the following: top-10 from 7 to 14; top-20 from 24 to 35; and top-40 from 65 to 86.

When analyzing the new traces recovered by all techniques, we noticed that bug report 1299458 (previously mentioned as one of the bug reports that left the group of traces not captured by any technique) had a true positive trace added to the group of traces captured by all techniques. This reinforces the *positive changes achieved through textual and information cleaning on this bug report*.

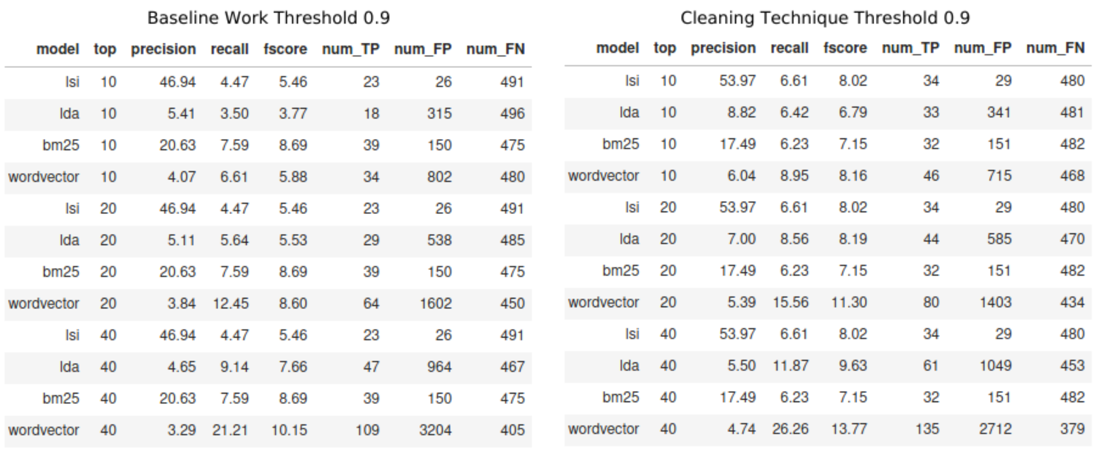
Despite the increase in the number of traces recovered by all techniques, a single bug report, 1432915, related to the *New Awesome Bar* system feature, has now missed traces (false negatives) by the LDA and Word Vector techniques. When analyzing it, we could notice the lack of textual information in its description, consisting of just two sentences. As the punctuation cleaning did not affect it, we can say that the information fields removal after the cleaning technique had a negative influence on the trace recovery process, in this case.

5.1.2 Second Scenario - Similarity Threshold 0.9

The scenario of threshold 0.9 favors precision over recall. In other words, in this scenario there is a larger number of false negatives traces.

Second Scenario General Result

In the second scenario, with the exception of BM25, all techniques improved in terms of precision, recall, F_2-Score, true and false positives and false negatives, as displayed in Figure 5.4.



Baseline Work Threshold 0.9

model	top	precision	recall	fscore	num_TP	num_FP	num_FN
lsi	10	46.94	4.47	5.46	23	26	491
lda	10	5.41	3.50	3.77	18	315	496
bm25	10	20.63	7.59	8.69	39	150	475
wordvector	10	4.07	6.61	5.88	34	802	480
lsi	20	46.94	4.47	5.46	23	26	491
lda	20	5.11	5.64	5.53	29	538	485
bm25	20	20.63	7.59	8.69	39	150	475
wordvector	20	3.84	12.45	8.60	64	1602	450
lsi	40	46.94	4.47	5.46	23	26	491
lda	40	4.65	9.14	7.66	47	964	467
bm25	40	20.63	7.59	8.69	39	150	475
wordvector	40	3.29	21.21	10.15	109	3204	405

Cleaning Technique Threshold 0.9

model	top	precision	recall	fscore	num_TP	num_FP	num_FN
lsi	10	53.97	6.61	8.02	34	29	480
lda	10	8.82	6.42	6.79	33	341	481
bm25	10	17.49	6.23	7.15	32	151	482
wordvector	10	6.04	8.95	8.16	46	715	468
lsi	20	53.97	6.61	8.02	34	29	480
lda	20	7.00	8.56	8.19	44	585	470
bm25	20	17.49	6.23	7.15	32	151	482
wordvector	20	5.39	15.56	11.30	80	1403	434
lsi	40	53.97	6.61	8.02	34	29	480
lda	40	5.50	11.87	9.63	61	1049	453
bm25	40	17.49	6.23	7.15	32	151	482
wordvector	40	4.74	26.26	13.77	135	2712	379

Figure 5.4: Baseline work and text and information results: Threshold 0.9 scenario.

Traces Missed by All Techniques

There is a decrease in the number of true positive traces not captured by all techniques, even with BM25 having a negative result in all cuts. The number of traces not captured by all techniques decreased from 361 to 333 in top-40.

Some traces related to bug report 1293308 that were recovered by the cleaning technique in the 0.0 threshold scenario were not recovered in the 0.9 scenario. This means that despite the positive influence of the cleaning technique on this bug report, the similarity scores did not reach 0.9.

The bug reports 1279143 and 1311998 (related to *New Awesome Bar* feature) are example of bug report that had traces removed from the list of traces missed by all techniques. The structure of these bug reports is similar, description with a lot of textual information. However, there are no important terms joined by punctuation, in these cases the improvement can be attributed to the removal of fields from the bug report with little relevant textual information.

Traces Recovered by All Techniques

This scenario has only three traces recovered by all techniques in the baseline work. These three traces are related to a single bug report, the bug report 1248267 (*Context Menu* feature). After applying the text and information cleaning, the number of traces recovered by all techniques increased to four. One trace related to the bug report 1345687 (related to *Indicator for device permissions* feature)was added to the set.

This bug report has some terms related to media devices, relevant terms for the traceability process. For example, in its description has the terms "camera", "microphone" merged by back slash, and "MediaRecorder" in an URL. The punctuation removal process is able to separate these terms from the other words, what increase their relevance in the query.

5.2 RQ2 - What is the effect of using an English spellchecker on the baseline work?

As we can see in Figure 5.5, the use of the spellchecker technique also achieved an improvement in all traceability techniques compared to the baseline study. In general, the result were not much different from the cleaning technique, both in qualitative and quantitative terms, although it was slightly better. The spellchecker did not find many wrong words (the vast majority were computing context words that the spellchecker did not recognize). So, the few typos contained in the used bug report dataset did not affect the trace recovery process. The improvement achieved through this approach can be attributed to the punctuation removal process, the same done in the textual and information cleaning technique.

The similarity between the results of this technique and the previous one shows us that textual cleaning (the process in common between them) has more effect on the process of recovering traceability than the removal of the bug report fields (present in the text and information cleaning) and the use of the spellchecker, in the study's general context.

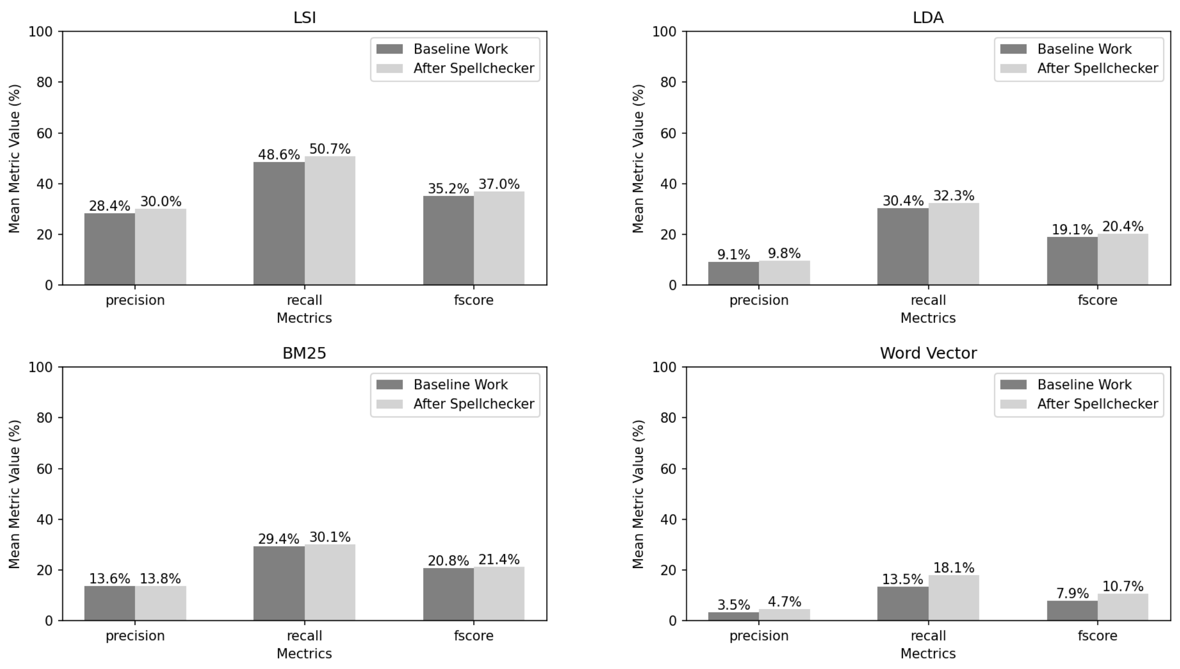
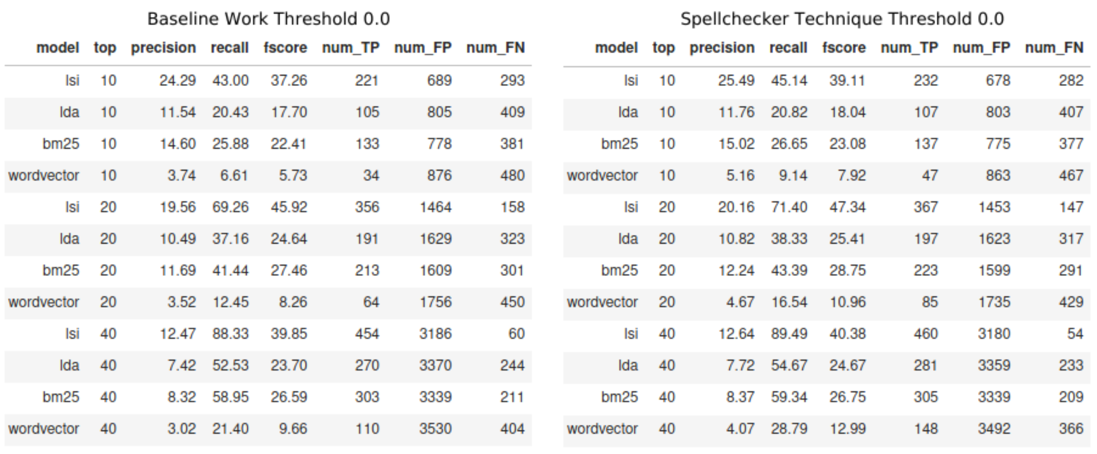


Figure 5.5: Baseline work and spellchecker technique general results.

5.2.1 First Scenario - Similarity Threshold 0.0

First Scenario General Result

Unlike the text and information cleaning technique, the spellchecker obtained positive results in all traceability techniques and in all cuts, as displayed in Figure 5.6.



Baseline Work Threshold 0.0

model	top	precision	recall	fscore	num_TP	num_FP	num_FN
lsi	10	24.29	43.00	37.26	221	689	293
lda	10	11.54	20.43	17.70	105	805	409
bm25	10	14.60	25.88	22.41	133	778	381
wordvector	10	3.74	6.61	5.73	34	876	480
lsi	20	19.56	69.26	45.92	356	1464	158
lda	20	10.49	37.16	24.64	191	1629	323
bm25	20	11.69	41.44	27.46	213	1609	301
wordvector	20	3.52	12.45	8.26	64	1756	450
lsi	40	12.47	88.33	39.85	454	3186	60
lda	40	7.42	52.53	23.70	270	3370	244
bm25	40	8.32	58.95	26.59	303	3339	211
wordvector	40	3.02	21.40	9.66	110	3530	404

Spellchecker Technique Threshold 0.0

model	top	precision	recall	fscore	num_TP	num_FP	num_FN
lsi	10	25.49	45.14	39.11	232	678	282
lda	10	11.76	20.82	18.04	107	803	407
bm25	10	15.02	26.65	23.08	137	775	377
wordvector	10	5.16	9.14	7.92	47	863	467
lsi	20	20.16	71.40	47.34	367	1453	147
lda	20	10.82	38.33	25.41	197	1623	317
bm25	20	12.24	43.39	28.75	223	1599	291
wordvector	20	4.67	16.54	10.96	85	1735	429
lsi	40	12.64	89.49	40.38	460	3180	54
lda	40	7.72	54.67	24.67	281	3359	233
bm25	40	8.37	59.34	26.75	305	3339	209
wordvector	40	4.07	28.79	12.99	148	3492	366

Figure 5.6: Baseline work and spellchecker results: Threshold 0.0 scenario.

	Traces Missed by All Techniques	
	Baseline	Spellchecker
Top 10	214	202
Top 20	104	86
Top 40	33	25

Table 5.3: Amount of traces missed by all techniques - Baseline work and Spellchecker.

Improvements in the result can be seen in the precision, recall, F_2-Score rates, and the amount of true and false positives, and false negatives. The detailed traces evaluation follows.

Traces Missed by All Techniques

The spellchecker improvement technique also achieved a decrease in the amount of correct traces not captured by any technique, as displayed in Table 5.3. In top-10, the number decreased from 214 to 202, from 104 to 86 in top-20, and from 33 to 25 in the top-40 cut.

Once again, the spellchecker technique behaves similarly to the cleaning technique. For instance, the bug reports 1293308 and 1299458, and the same traces, related to the *Context Menu* system feature, also left traces not captured by any technique in the top-40 cut, in the same way as the cleaning technique.

Traces Recovered by All Techniques

There was also an improvement in the number of traces recovered by all techniques. Comparing to the baseline study, we have an increase from 7 to 11 in top-10, from 24 to 32 in top-20, and from 65 to 88 in top-40, as shown in Table 5.4. The results obtained in the Top 40 cut slightly overcome the text and information cleaning technique. For instance, bug report 1287384 had the spellchecker technique able to recover two additional traces for all techniques. The spellchecker technique had a different bug report added to the list of traces captured by all techniques (when comparing with the baseline study and the cleaning technique), the bug report 1311998. This is a well-written bug report, containing relevant context words like “Awesomebar”. However there are no typos, what once again shows that the spelling errors hindering the traceability process do not apply to this bug reports dataset.

	Traces Recovered by All Techniques	
	Baseline	Spellchecker
Top 10	7	11
Top 20	24	32
Top 40	65	88

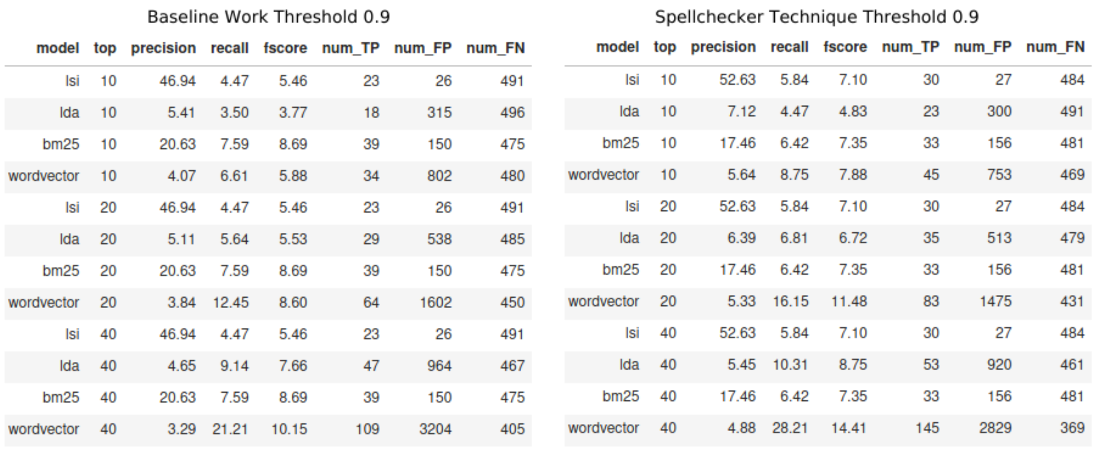
Table 5.4: Amount of traces recovered by all techniques - Baseline work and Spellchecker.

We believe that these traces have been recovered due to the removal of punctuation, but without removing the information fields as occurs in the cleaning technique.

5.2.2 Second Scenario - Similarity Threshold 0.9

Second Scenario General Result

Figure 5.7 display the result obtained by the spellchecker technique in the 0.9 threshold scenario. Like the cleaning technique, there is a negative result for the BM25 technique, in all cuts. But for all other traceability techniques there is a slight improvement.



Baseline Work Threshold 0.9

model	top	precision	recall	fscore	num_TP	num_FP	num_FN
lsi	10	46.94	4.47	5.46	23	26	491
lda	10	5.41	3.50	3.77	18	315	496
bm25	10	20.63	7.59	8.69	39	150	475
wordvector	10	4.07	6.61	5.88	34	802	480
lsi	20	46.94	4.47	5.46	23	26	491
lda	20	5.11	5.64	5.53	29	538	485
bm25	20	20.63	7.59	8.69	39	150	475
wordvector	20	3.84	12.45	8.60	64	1602	450
lsi	40	46.94	4.47	5.46	23	26	491
lda	40	4.65	9.14	7.66	47	964	467
bm25	40	20.63	7.59	8.69	39	150	475
wordvector	40	3.29	21.21	10.15	109	3204	405

Spellchecker Technique Threshold 0.9

model	top	precision	recall	fscore	num_TP	num_FP	num_FN
lsi	10	52.63	5.84	7.10	30	27	484
lda	10	7.12	4.47	4.83	23	300	491
bm25	10	17.46	6.42	7.35	33	156	481
wordvector	10	5.64	8.75	7.88	45	753	469
lsi	20	52.63	5.84	7.10	30	27	484
lda	20	6.39	6.81	6.72	35	513	479
bm25	20	17.46	6.42	7.35	33	156	481
wordvector	20	5.33	16.15	11.48	83	1475	431
lsi	40	52.63	5.84	7.10	30	27	484
lda	40	5.45	10.31	8.75	53	920	461
bm25	40	17.46	6.42	7.35	33	156	481
wordvector	40	4.88	28.21	14.41	145	2829	369

Figure 5.7: Baseline work and spellchecker results: Threshold 0.9 scenario.

Traces Missed by All Techniques

In the 0.9 threshold scenario the amount of traces missed by all techniques decreased from 361 to 328, in top-40, after applying the spellchecker technique.

The bug report 1296322 (related to *Indicator for device permissions*) is a example of a trace that leaves the set of traces missed by all technique after applying the spellchecker technique. Another example where the spelling correction process itself had no weight at all, we can attribute this change to removing punctuation. In this bug report description has some words relevant to the feature merged by slash, such as "Info/Permissions" and "Allow/Block".

Traces Recovered by All Techniques

The same trace and bug report (1345687) added in the cleaning technique was also added after the application of the spellchecker technique. In terms of traces recovered by all techniques the result was the same as the previous technique.

5.3 RQ3 - What is the effect of doubling the bug report title or description on the baseline work?

5.3.1 Doubling the Title

The duplication of the bug reports title field achieved a small improvement, as we can see in Figure 5.8. Among the improvement attempts addressed so far, this one has reached the lowest performance.

LSI technique had a increase in precision of 0.6%, 0.7% in recall, and 0.7% in F_2-Score; LDA increased its precision in 0.6%, recall in 0.7%, and the F_2-Score in 0.8%; the BM25 technique had a increase of 0.1% in precision, 0.6% in recall, and 0.5% in F_2-Score; Word Vector had the precision increased in 0.3%, the recall in 1.2%, and the F_2-Score in 0.7%.

First Scenario - Similarity Threshold 0.0

First Scenario General Result

Despite the general positive result, in the scenario of threshold 0.0 the title duplication technique had a negative performance for the LSI and LDA in the top-40. For the other traceability techniques in all other cuts, the results are positive, as can be seen in Figure 5.9.

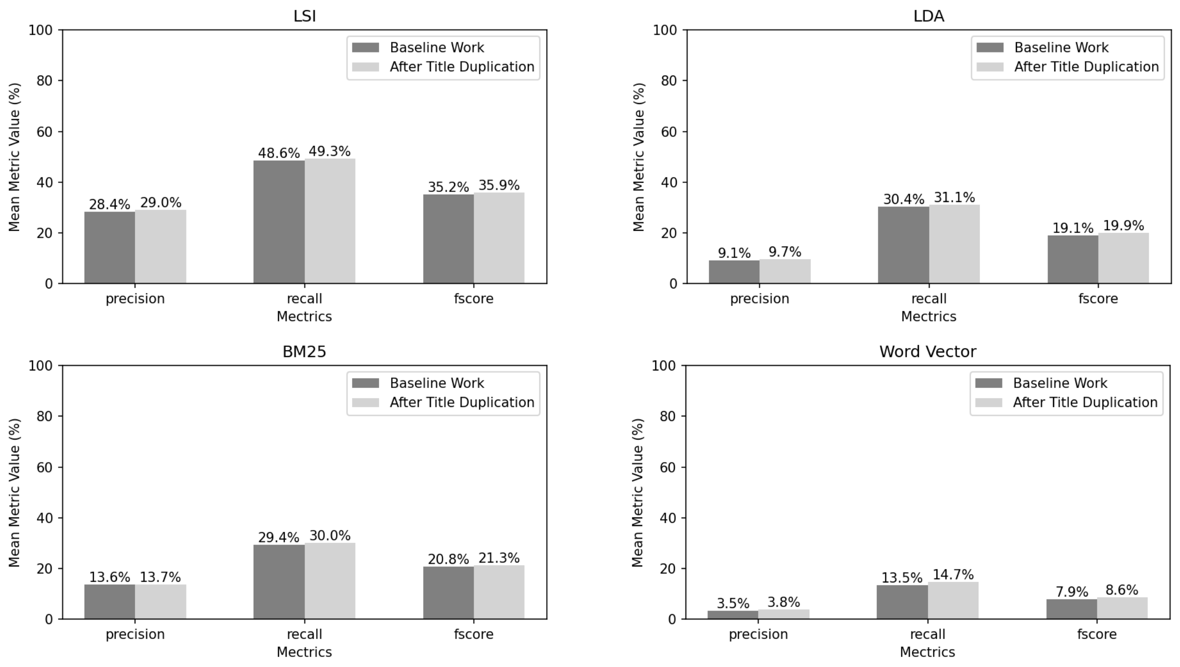
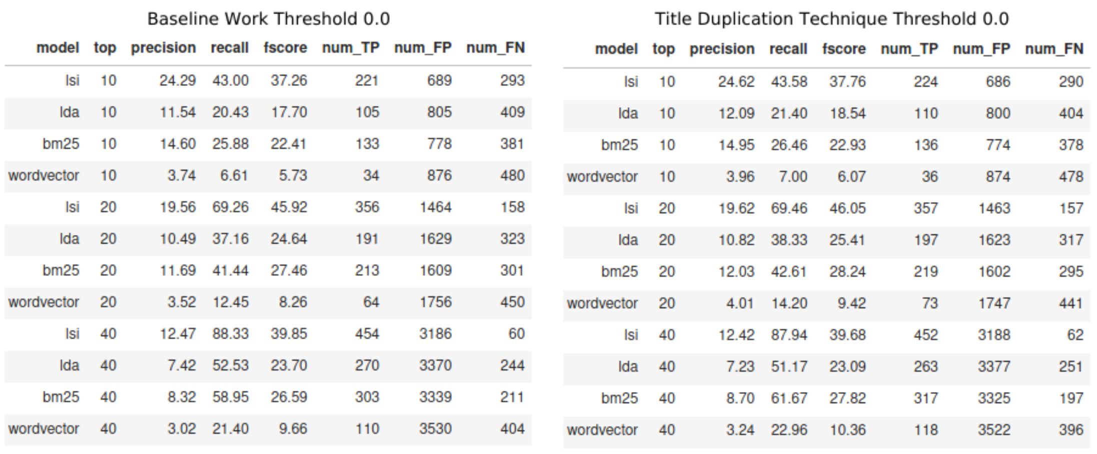


Figure 5.8: Baseline work and title duplication technique general results.



Baseline Work Threshold 0.0

model	top	precision	recall	fscore	num_TP	num_FP	num_FN
lsi	10	24.29	43.00	37.26	221	689	293
lda	10	11.54	20.43	17.70	105	805	409
bm25	10	14.60	25.88	22.41	133	778	381
wordvector	10	3.74	6.61	5.73	34	876	480
lsi	20	19.56	69.26	45.92	356	1464	158
lda	20	10.49	37.16	24.64	191	1629	323
bm25	20	11.69	41.44	27.46	213	1609	301
wordvector	20	3.52	12.45	8.26	64	1756	450
lsi	40	12.47	88.33	39.85	454	3186	60
lda	40	7.42	52.53	23.70	270	3370	244
bm25	40	8.32	58.95	26.59	303	3339	211
wordvector	40	3.02	21.40	9.66	110	3530	404

Title Duplication Technique Threshold 0.0

model	top	precision	recall	fscore	num_TP	num_FP	num_FN
lsi	10	24.62	43.58	37.76	224	686	290
lda	10	12.09	21.40	18.54	110	800	404
bm25	10	14.95	26.46	22.93	136	774	378
wordvector	10	3.96	7.00	6.07	36	874	478
lsi	20	19.62	69.46	46.05	357	1463	157
lda	20	10.82	38.33	25.41	197	1623	317
bm25	20	12.03	42.61	28.24	219	1602	295
wordvector	20	4.01	14.20	9.42	73	1747	441
lsi	40	12.42	87.94	39.68	452	3188	62
lda	40	7.23	51.17	23.09	263	3377	251
bm25	40	8.70	61.67	27.82	317	3325	197
wordvector	40	3.24	22.96	10.36	118	3522	396

Figure 5.9: Baseline work and title duplication results: Threshold 0.0 scenario.

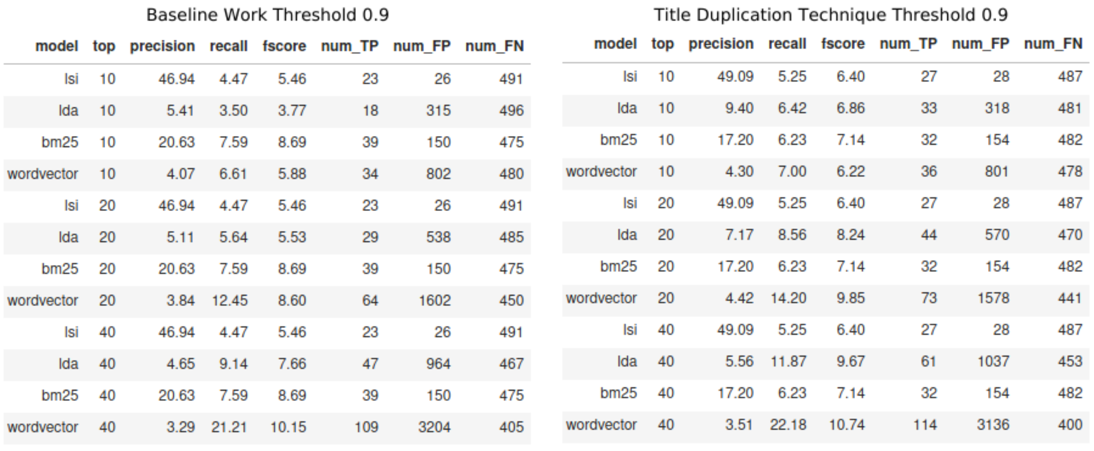
Traces Missed by All Techniques The number of true positives traces not captured by any improvement technique had a small decrease, from 214 to 212 in the top-10 and from 104 to 101 in top-20. Despite the overall result's improvement, the number of correct traces not captured increased from 33 to 35 in the top-40 cut.

All the six bug reports from the baseline work list of traces missed by all the traceability techniques were also missed after applying the double title technique, and a new bug report was added in the top-40 cut. The bug report 1270983 (related to the *Context Menu* feature) had five traces added in the list of traces missed by all techniques. This bug report has only a link as a description. The title is "Intermittent browser_ contextmenu.js | Test timed out | Found a tab after the previous test timed out: subtst_contextmenu.html -", and even containing the word "contextmenu", the traces were lost after the title duplication.

To exemplify a positive change, the bug report 1293308 had two true positives traces now captured by at least one technique, after doubling the title field. Unlike bug report 1270983, this report has a detailed description. Its title ("System font setting respected by location bar but ignored by location bar's autocomplete drop-down list") contains relevant words, written more cleanly.

Traces Recovered by All Techniques There was also a slight increase in the number of correct traces captured by every technique in all cuts, from 7 to 8 in top-10 cut, from 24 to 28 in top-20 and from 65 to 72 in top-40 cut.

One of these new traces is related to bug report 1357458 "After Customization - typed text in the Awesome bar doesn't correspond with the text from One-Off-Searches bar", related to the *Awesome Bar* feature. Its description contains the term "URL bar" used in many bug reports to specify the *Awesome Bar*. Sometimes, this term is not enough to generate a similarity rate with enough value to be recovered as a trace. However, when duplicating relevant words as "searches bar" and "Awesome bar" present in the title, we increase their relevance in recovering traces.



Baseline Work Threshold 0.9

model	top	precision	recall	fscore	num_TP	num_FP	num_FN
lsi	10	46.94	4.47	5.46	23	26	491
lda	10	5.41	3.50	3.77	18	315	496
bm25	10	20.63	7.59	8.69	39	150	475
wordvector	10	4.07	6.61	5.88	34	802	480
lsi	20	46.94	4.47	5.46	23	26	491
lda	20	5.11	5.64	5.53	29	538	485
bm25	20	20.63	7.59	8.69	39	150	475
wordvector	20	3.84	12.45	8.60	64	1602	450
lsi	40	46.94	4.47	5.46	23	26	491
lda	40	4.65	9.14	7.66	47	964	467
bm25	40	20.63	7.59	8.69	39	150	475
wordvector	40	3.29	21.21	10.15	109	3204	405

Title Duplication Technique Threshold 0.9

model	top	precision	recall	fscore	num_TP	num_FP	num_FN
lsi	10	49.09	5.25	6.40	27	28	487
lda	10	9.40	6.42	6.86	33	318	481
bm25	10	17.20	6.23	7.14	32	154	482
wordvector	10	4.30	7.00	6.22	36	801	478
lsi	20	49.09	5.25	6.40	27	28	487
lda	20	7.17	8.56	8.24	44	570	470
bm25	20	17.20	6.23	7.14	32	154	482
wordvector	20	4.42	14.20	9.85	73	1578	441
lsi	40	49.09	5.25	6.40	27	28	487
lda	40	5.56	11.87	9.67	61	1037	453
bm25	40	17.20	6.23	7.14	32	154	482
wordvector	40	3.51	22.18	10.74	114	3136	400

Figure 5.10: Baseline work and title duplication results: Threshold 0.9 scenario.

5.3.2 Second Scenario - Similarity Threshold 0.9

Second Scenario General Result

Traces Missed by All Techniques The number of traces missed by all techniques decreased from 361 to 351 in top-40 cut. Repeating the behavior of scenario 0.0, the technique of duplicating the title obtained the lowest result among the improvement techniques evaluated so far.

The bug report 1296861 (*Indicator for device permissions*) and 1305195 (*Zoom Indicator* feature) are examples of bug reports that had traces removed from the list of traces not captured by all techniques. In these two cases there is the presence of terms relevant to the process of recovering traces, duplicating the title increased the relevance of these terms in the query.

Traces Recovered by All Techniques Despite the positive result, there is a decrease in the number of traces recovered by all traceability techniques. The number dropped from three to two. These traces are related to a single bug report, the bug report 1248267.

5.3.3 Doubling the Description

Doubling the bug report description field achieved a negative performance in all evaluations, as shown in Figure 5.11. Of all the improvement techniques evaluated so far, this was the only one to have a negative result. This is due to the fact that in the description of the bug reports there are usually more terms that are not important for the traceability recovery process, and by duplicating the description, the terms that are actually important lose more relevance within the query.

LSI had a decrease of 2.5% in precision, 3.3% in recall, and 2.5% in F_2-Score; the LDA technique decreased 1% in precision, 3.9% in recall, and 2.3% in F_2-Score; the BM25 technique had a decrease of 1.3% in precision, 2.8% in recall, and 2% in F_2-Score; and the Word Vector technique precision decreases 0.2%, the recall decreases 0.7%, and the F_2-Score 0.4%.

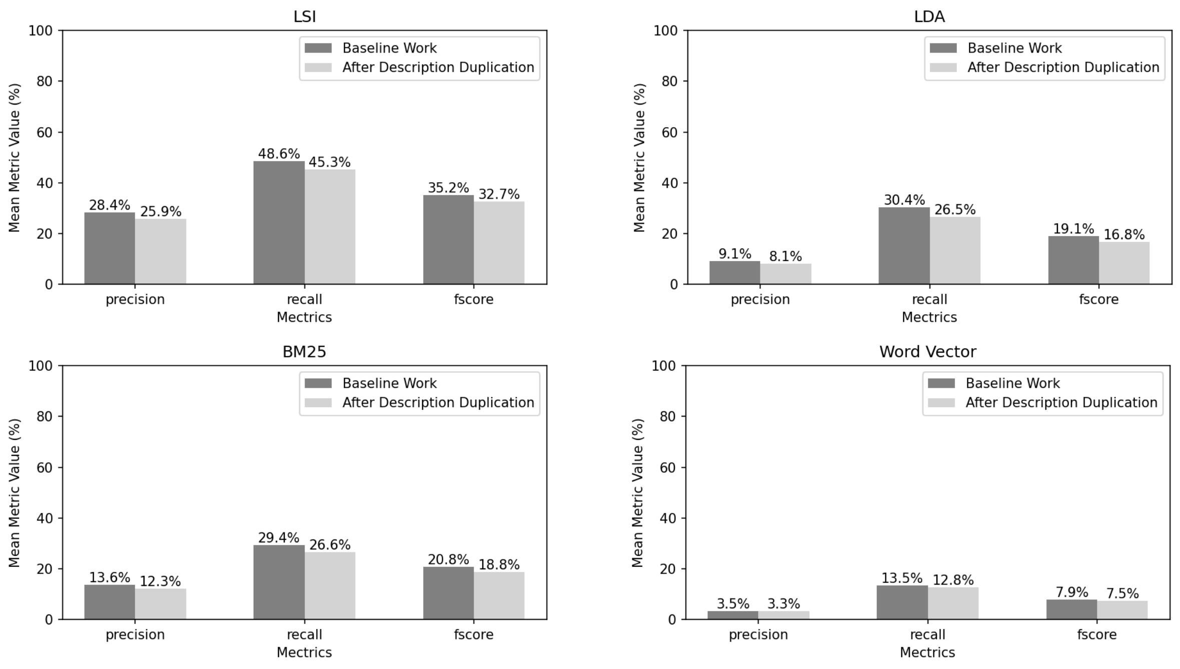


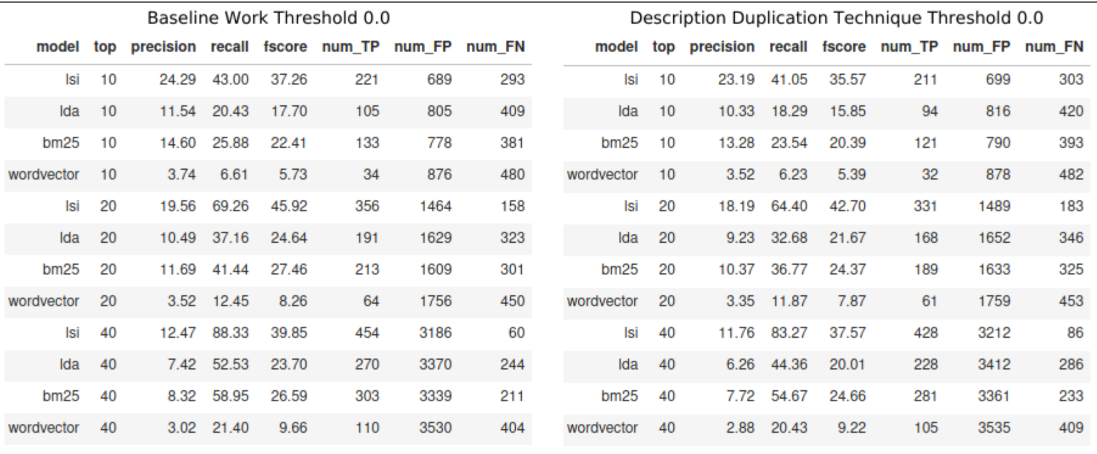
Figure 5.11: Baseline work and description duplication technique general results.

First Scenario - Similarity Threshold 0.0

First Scenario General Result

Following the general result of the technique, the result in scenario 0.0 threshold is also negative. For all techniques, in all cuts, both in terms of precision, recall and F_2-Score, and amount of true and false positives and false negatives. The result is displayed in Figure 5.12.

Traces Missed by All Techniques The amount of missed true positive traces has increased from 214 to 226 in top-10, from 104 to 123 in top-20 cut and from 33 to 51 in top-40.



Baseline Work Threshold 0.0

model	top	precision	recall	fscore	num_TP	num_FP	num_FN
lsi	10	24.29	43.00	37.26	221	689	293
lda	10	11.54	20.43	17.70	105	805	409
bm25	10	14.60	25.88	22.41	133	778	381
wordvector	10	3.74	6.61	5.73	34	876	480
lsi	20	19.56	69.26	45.92	356	1464	158
lda	20	10.49	37.16	24.64	191	1629	323
bm25	20	11.69	41.44	27.46	213	1609	301
wordvector	20	3.52	12.45	8.26	64	1756	450
lsi	40	12.47	88.33	39.85	454	3186	60
lda	40	7.42	52.53	23.70	270	3370	244
bm25	40	8.32	58.95	26.59	303	3339	211
wordvector	40	3.02	21.40	9.66	110	3530	404

Description Duplication Technique Threshold 0.0

model	top	precision	recall	fscore	num_TP	num_FP	num_FN
lsi	10	23.19	41.05	35.57	211	699	303
lda	10	10.33	18.29	15.85	94	816	420
bm25	10	13.28	23.54	20.39	121	790	393
wordvector	10	3.52	6.23	5.39	32	878	482
lsi	20	18.19	64.40	42.70	331	1489	183
lda	20	9.23	32.68	21.67	168	1652	346
bm25	20	10.37	36.77	24.37	189	1633	325
wordvector	20	3.35	11.87	7.87	61	1759	453
lsi	40	11.76	83.27	37.57	428	3212	86
lda	40	6.26	44.36	20.01	228	3412	286
bm25	40	7.72	54.67	24.66	281	3361	233
wordvector	40	2.88	20.43	9.22	105	3535	409

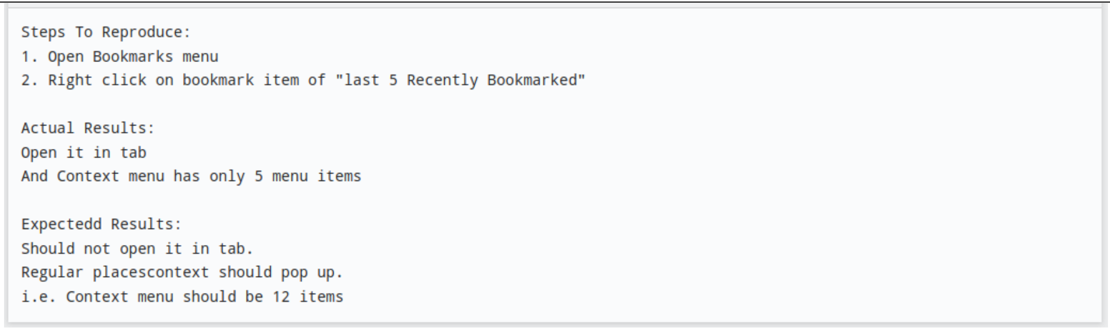
Figure 5.12: Baseline work and description duplication results: Threshold 0.0 scenario.

After doubling the description field the traceability techniques missed correct traces related to the *Indicator for device permissions* feature, a feature that had no traces in the list of true positives traces missed by all techniques in the baseline work. We observed that most terms in descriptions are irrelevant for traceability, and duplicating them further decreases the relevance of essential terms.

To exemplify the poor performance of this technique, bug report 1264988, "Scrollbar appears for a moment in the new Awesome bar Resultlist", had no traces added to the list of traces not captured by any of the traceability techniques in the baseline work. This case is compelling, as its title has the name of the feature to which it relates, the Awesome Bar, which is accurate for the traceability process. However, in its description the feature is cited as "Location Bar", and by duplicating the bug report description in the query, the correct term "Awesombar" present in the title loses relevance in traceability.

Traces Recovered by All Techniques There was no change in the number of traces captured by all techniques in the top-10 cut. In top-20, the number decreased from 24 to 23, and 65 to 59 in top-40.

Despite the poor overall performance, there were still new true positives traces recovered by all techniques, as the bug report 1248267. The description of this bug report is short, but with several relevant terms, as displayed in Figure 5.13.



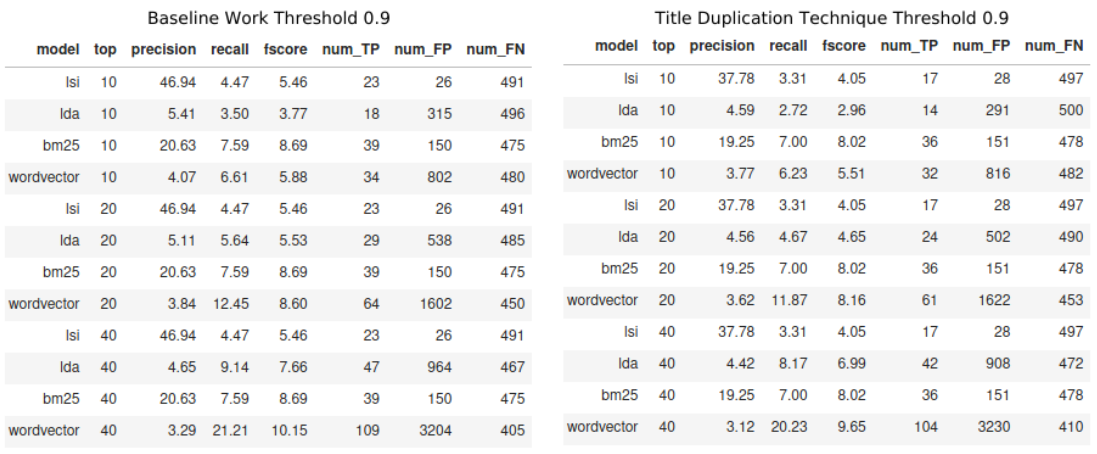
```
Steps To Reproduce:
1. Open Bookmarks menu
2. Right click on bookmark item of "last 5 Recently Bookmarked"

Actual Results:
Open it in tab
And Context menu has only 5 menu items

Expectedd Results:
Should not open it in tab.
Regular placescontext should pop up.
i.e. Context menu should be 12 items
```

Figure 5.13: Bug report 1248267 description.

5.3.4 Second Scenario - Similarity Threshold 0.9

Second Scenario General Result

As expected, the description duplication technique also had a negative result in the threshold 0.9 scenario. As can be seen in Figure 5.14, there is a slight worsening in the rates of precision, recall, and F_2-Score, and in the amount of true and false positives, and false negatives.



Baseline Work Threshold 0.9

model	top	precision	recall	fscore	num_TP	num_FP	num_FN
lsi	10	46.94	4.47	5.46	23	26	491
lda	10	5.41	3.50	3.77	18	315	496
bm25	10	20.63	7.59	8.69	39	150	475
wordvector	10	4.07	6.61	5.88	34	802	480
lsi	20	46.94	4.47	5.46	23	26	491
lda	20	5.11	5.64	5.53	29	538	485
bm25	20	20.63	7.59	8.69	39	150	475
wordvector	20	3.84	12.45	8.60	64	1602	450
lsi	40	46.94	4.47	5.46	23	26	491
lda	40	4.65	9.14	7.66	47	964	467
bm25	40	20.63	7.59	8.69	39	150	475
wordvector	40	3.29	21.21	10.15	109	3204	405

Title Duplication Technique Threshold 0.9

model	top	precision	recall	fscore	num_TP	num_FP	num_FN
lsi	10	37.78	3.31	4.05	17	28	497
lda	10	4.59	2.72	2.96	14	291	500
bm25	10	19.25	7.00	8.02	36	151	478
wordvector	10	3.77	6.23	5.51	32	816	482
lsi	20	37.78	3.31	4.05	17	28	497
lda	20	4.56	4.67	4.65	24	502	490
bm25	20	19.25	7.00	8.02	36	151	478
wordvector	20	3.62	11.87	8.16	61	1622	453
lsi	40	37.78	3.31	4.05	17	28	497
lda	40	4.42	8.17	6.99	42	908	472
bm25	40	19.25	7.00	8.02	36	151	478
wordvector	40	3.12	20.23	9.65	104	3230	410

Figure 5.14: Baseline work and description duplication results: Threshold 0.9 scenario.

Traces Missed by All Techniques The number of true positives missed by all techniques increased from 361 to 375. The reason is the same as in scenario 0.0, duplicating the description generally decreases the relevance of important terms in the query, it would not be different in a scenario that values precision more (0.9 threshold scenario).

Traces Recovered by All Techniques There is no change in true positives traces recovered by all techniques. The same three traces related to the single bug report 1248267. This bug report is the same one that had traces recovered by duplicating the description in the 0.0 scenario.

5.4 RQ4 - What is the effect of combining the three improvement techniques with positive results (cleaning, spellchecker, and doubling the title)?

As the description duplication had negative results, we excluded it from the combination of techniques. The combination of the three improvement techniques with positive results surpassed the results of each separately, with higher precision, recall, and F_2-Score. The results are displayed in Figure 5.15. Although some results of applying an approach in isolation have demonstrated small improvements, their combination reached a more comprehensive gain because the sets of recovered traces (by applying each one) have some disjoint traces, *i.e.* they recover some different positive trace links.

In LSI technique the precision increase 2%, the recall 3.4%, and the F_2-Score 3.1%; LDA had its precision increased by 1%, the recall by 2.3%, and the F_2-Score 1.7%; BM25 had also 1% of increase in precision, 1.7% in recall, and 1.5% in F_2-Score; the Word Vector technique had an increase of 1.8% in precision, 6.4% in recall, and 4% in F_2-Score.

5.4.1 First Scenario - Similarity Threshold 0.0

First Scenario General Result

The first scenario result displayed in Figure 5.16 follow the general results of the techniques combination. Overcoming the title duplication in all traceability techniques and cuts, and the cleaning and spellchecker techniques in almost all traceability techniques and cuts.

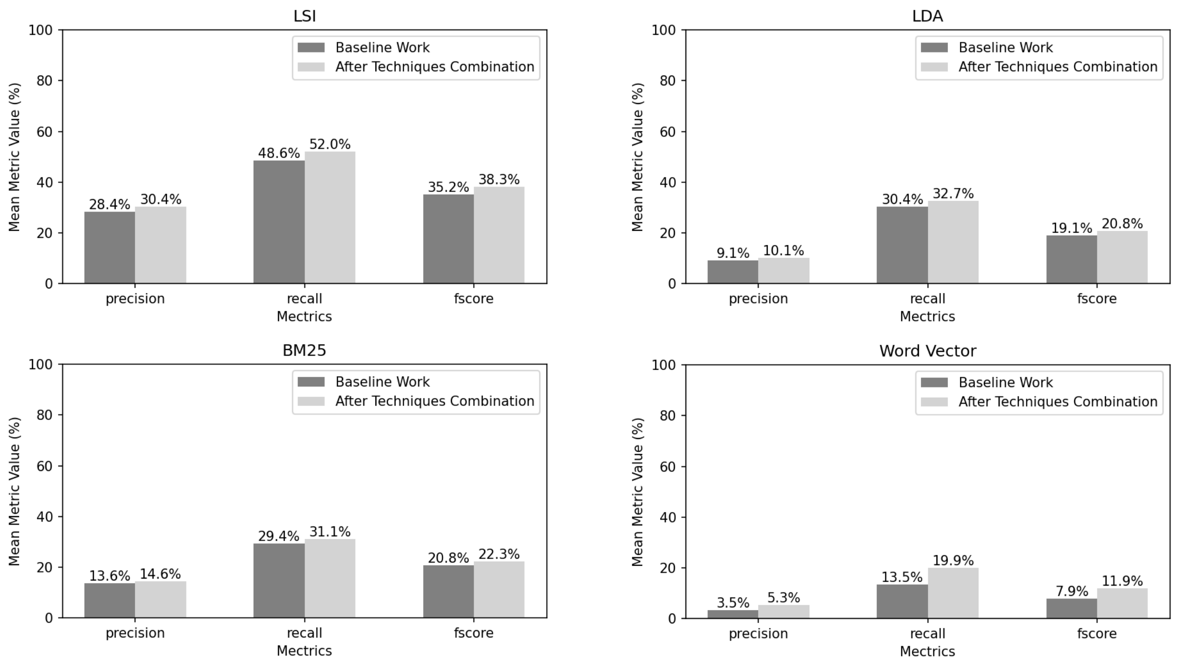
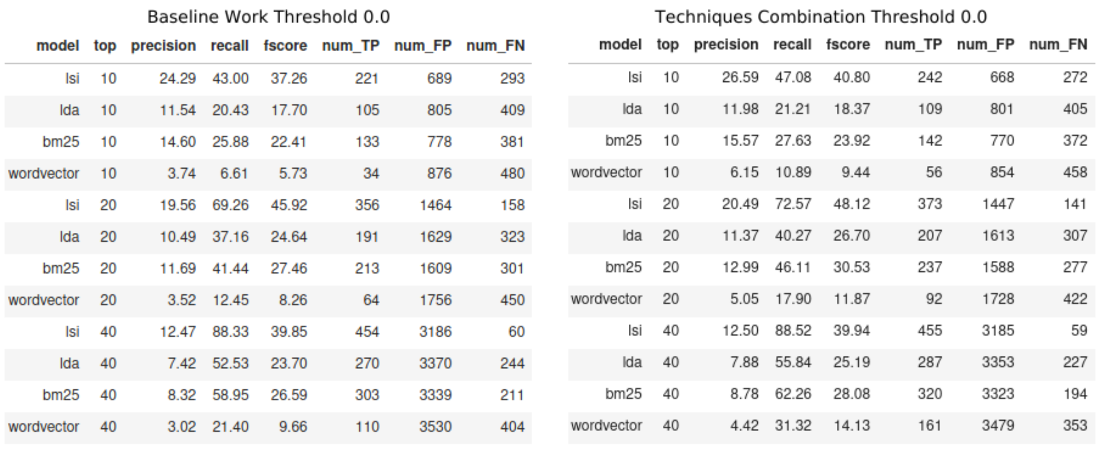


Figure 5.15: Baseline work and techniques combination general results.



Baseline Work Threshold 0.0

model	top	precision	recall	fscore	num_TP	num_FP	num_FN
lsi	10	24.29	43.00	37.26	221	689	293
lda	10	11.54	20.43	17.70	105	805	409
bm25	10	14.60	25.88	22.41	133	778	381
wordvector	10	3.74	6.61	5.73	34	876	480
lsi	20	19.56	69.26	45.92	356	1464	158
lda	20	10.49	37.16	24.64	191	1629	323
bm25	20	11.69	41.44	27.46	213	1609	301
wordvector	20	3.52	12.45	8.26	64	1756	450
lsi	40	12.47	88.33	39.85	454	3186	60
lda	40	7.42	52.53	23.70	270	3370	244
bm25	40	8.32	58.95	26.59	303	3339	211
wordvector	40	3.02	21.40	9.66	110	3530	404

Techniques Combination Threshold 0.0

model	top	precision	recall	fscore	num_TP	num_FP	num_FN
lsi	10	26.59	47.08	40.80	242	668	272
lda	10	11.98	21.21	18.37	109	801	405
bm25	10	15.57	27.63	23.92	142	770	372
wordvector	10	6.15	10.89	9.44	56	854	458
lsi	20	20.49	72.57	48.12	373	1447	141
lda	20	11.37	40.27	26.70	207	1613	307
bm25	20	12.99	46.11	30.53	237	1588	277
wordvector	20	5.05	17.90	11.87	92	1728	422
lsi	40	12.50	88.52	39.94	455	3185	59
lda	40	7.88	55.84	25.19	287	3353	227
bm25	40	8.78	62.26	28.08	320	3323	194
wordvector	40	4.42	31.32	14.13	161	3479	353

Figure 5.16: Baseline work and techniques combination results: Threshold 0.0 scenario.

	Traces Missed by All Techniques	
	Baseline	Techniques Combination
Top 10	214	202
Top 20	104	86
Top 40	33	25

Table 5.5: Amount of traces missed by all techniques - Baseline work and Techniques combination.

Traces Missed by All Techniques

Compared to the baseline study, the amount of true positive traces missed by any technique decreased in all cuts, as displayed in Table 5.5. Top-10 decreased from 214 to 199, from 104 to 82 in the top-20 and from 33 to 24 in top-40. When analyzing these traces, we noticed the repetition of patterns of the three combined techniques. For instance, bug reports 1293308 and 1299458 are now detected by at least one technique, in the same way that occurred in the text and information cleaning and the spellchecker techniques. However, this also happened to the negative results. For bug reports 1279864 and 1270983, the first was missed by all traceability techniques both in information cleaning and spellchecker techniques. The double title technique added five true positives traces to the list o traces missed by all techniques when applied separately, in the improvement techniques combination only two traces related to this bug report were added.

Traces Recovered by All Techniques

There was an increase in the number of true positives traces captured by all techniques compared to the baseline study. As we can see in Table 5.6, in top-10 cut, the amount increased from 7 to 17, in top-20 from 24 to 36, and from 65 to 96 in top-40.

Bug report 1357458, which was added to the list in the title duplication technique, was also added in the combination. Besides that, four others bug reports (1297976, 1299458, 1296322, and 1296861) were added to the true positives traces captured by all techniques, repeating the pattern of the text and information cleaning and spellchecker techniques.

There is also a repetition of negative patterns in the set of traces recovered by all techniques. For instance, the bug report 1432915 left the list, just as it happened in the text and

	Traces Recovered by All Techniques	
	Baseline	Techniques Combination
Top 10	7	11
Top 20	24	32
Top 40	65	88

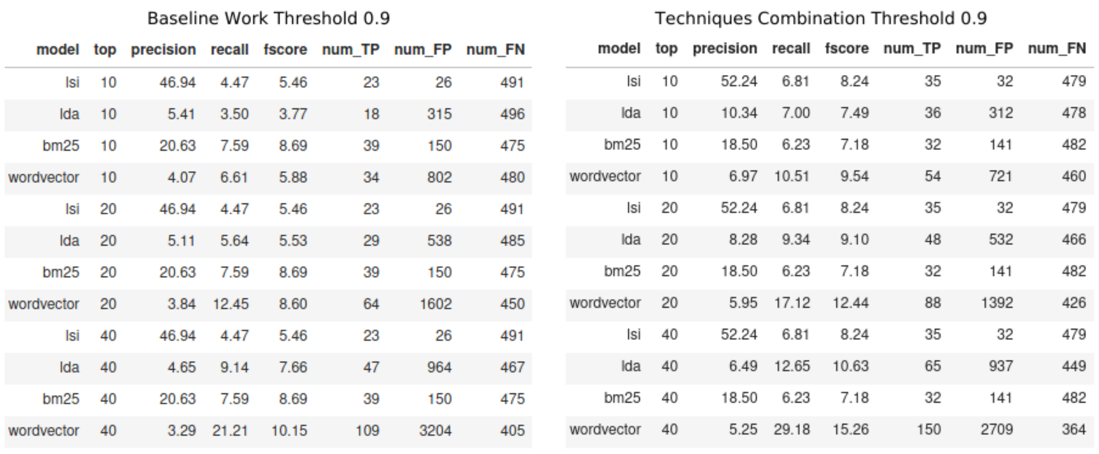
Table 5.6: Amount of traces recovered by all techniques - Baseline work and Techniques combination.

information cleaning and spellchecker techniques.

5.4.2 Second Scenario - Similarity Threshold 0.9

Second Scenario General Result

Even with the combination of improvement techniques, BM25 had a negative result, in the same way as the previous results in 0.9 scenario. But as shown in Figure 5.17, there is an improvement in the other traceability techniques, in all cuts.



Baseline Work Threshold 0.9

model	top	precision	recall	fscore	num_TP	num_FP	num_FN
lsi	10	46.94	4.47	5.46	23	26	491
lda	10	5.41	3.50	3.77	18	315	496
bm25	10	20.63	7.59	8.69	39	150	475
wordvector	10	4.07	6.61	5.88	34	802	480
lsi	20	46.94	4.47	5.46	23	26	491
lda	20	5.11	5.64	5.53	29	538	485
bm25	20	20.63	7.59	8.69	39	150	475
wordvector	20	3.84	12.45	8.60	64	1602	450
lsi	40	46.94	4.47	5.46	23	26	491
lda	40	4.65	9.14	7.66	47	964	467
bm25	40	20.63	7.59	8.69	39	150	475
wordvector	40	3.29	21.21	10.15	109	3204	405

Techniques Combination Threshold 0.9

model	top	precision	recall	fscore	num_TP	num_FP	num_FN
lsi	10	52.24	6.81	8.24	35	32	479
lda	10	10.34	7.00	7.49	36	312	478
bm25	10	18.50	6.23	7.18	32	141	482
wordvector	10	6.97	10.51	9.54	54	721	460
lsi	20	52.24	6.81	8.24	35	32	479
lda	20	8.28	9.34	9.10	48	532	466
bm25	20	18.50	6.23	7.18	32	141	482
wordvector	20	5.95	17.12	12.44	88	1392	426
lsi	40	52.24	6.81	8.24	35	32	479
lda	40	6.49	12.65	10.63	65	937	449
bm25	40	18.50	6.23	7.18	32	141	482
wordvector	40	5.25	29.18	15.26	150	2709	364

Figure 5.17: Baseline work and techniques combination results: Threshold 0.9 scenario.

Traces Missed by All Techniques

The amount of true positives traces missed by all techniques decreased from 361 to 319. As expected, the combination of techniques is able to recover some of the traces recovered by the techniques of cleaning, spellchecker, and title duplication in the 0.9 threshold scenario. Such as the traces related to the bug report 1311998, 1296322 and 1296861.

Traces Recovered by All Techniques

The number of true traces recovered by all techniques increased from three to four, the same trace (related to bug report 1345687) added by the cleaning and spellchecker techniques was added in the techniques combination. So, the combination of techniques is able to unite the positive results of improvement techniques in this scenario as well.

5.5 RQ5 - What are the results obtained by the hybrid techniques?

5.5.1 Similarity Matrices Merge

The similarity matrices merge technique creates a similarity matrix by merging the similarity matrices of each of the traceability techniques, the idea is to ensure that each of the traceability techniques can influence the similarity value of each trace. As previously described, the similarity values contained in the created matrix will be the weighted mean of the similarity values present in the matrices of traceability techniques. The better the result of a traceability technique in the baseline work, in terms of F_2-Score, the greater its weight in the creation of the new similarity matrix and greater its influence.

The result compared with the baseline work in the 0.0 threshold scenario is shown in Table 5.7 (top-40 cut). Although LSI is the technique with the greatest weight, the result obtained was lower than the LSI technique in the baseline study. While in the baseline study the LSI has 454 true positives traces in the top-40, with threshold value 0.0, the number of true positives achieved in the merged similarity matrix was 402 (in the same scenario). Comparing to the LSI technique (best result), the similarity matrices merge obtained a 1.43% lower result in terms of precision, 10.12% lower in recall, and 4.56% lower in terms of F_2-Score.

Despite the similarity matrices merge having achieved a better result than LDA, BM25, and Word Vector in the 0.0 threshold scenario, it failed to achieve its objective: uniting the

	Precision	Recall	F-Score	TP	FP	FN
S. M. Merge	11.04%	78.21%	35.29%	402	3238	112
LSI	12.47%	88.33%	39.85%	454	3186	60
LDA	7.42%	52.53%	23.70%	270	3370	244
BM25	8.32%	58.95%	26.59%	303	3339	211
Word Vector	3.02%	21.40%	9.66%	110	3530	404

Table 5.7: Similarity Matrices Merge results: 0.0 threshold top-40.

	Precision	Recall	F-Score	TP	FP	FN
S. M. Merge	38.89%	1.36%	1.69%	7	11	507
LSI	46.94%	4.47%	5.46%	23	26	491
LDA	4.65%	9.14%	7.66%	47	964	467
BM25	20.63%	7.56%	8.69%	39	150	475
Word Vector	3.29%	21.21%	10.15%	109	3204	405

Table 5.8: Similarity Matrices Merge results: 0.9 threshold top-40.

efforts of the four traceability techniques, not surpassing the LSI without no improvement applied.

The results obtained in 0.9 scenario, displayed in Table 5.8 (top-40), were worse than those in the first scenario. The limit value of 0.9 favors precision as it is more difficult to achieve. As in the previous scenario, even with LSI having greater weight in the production of the new similarity matrix, the final values were still below what they were in the base work. As a result we have the worst rates of precision, recall and F_2-Score, among the techniques for this scenario. LSI had the better result in precision. Compared to it, the similarity matrices merge obtained a 8.05% lower precision. In terms of recall and F_2-Score, word vector obtained the best results, comparing to word vector, the similarity matrices merge obtained a 19.85% lower recall value and a 8.46% lower result in terms of F_2-Score.

The weights are applied on the scores of the similarity matrices produced by each one of the four IR or ML techniques adopted in the baseline study. However, the weight attributed to the similarity scores of the LSI was not sufficient to recover all the traces originally recovered by it. In other words, the weights of the techniques that had the inferior results pulled the

Strategy	Threshold	Precision	Recall	F-Score	TP	FP	FN
At Least One	0.0	5.16%	93.58%	21.13%	481	8843	33
Tech Traces	0.9	3.66%	29.77%	12.27%	153	4024	361
At Least Two	0.0	10.15%	72.37%	32.52%	372	3292	142
Tech Traces	0.9	15.49%	10.70%	11.41%	55	300	459
At Least Three	0.0	16.87%	42.61%	32.65%	219	1079	295
Tech Traces	0.9	29.17%	1.36%	1.68%	7	17	507

Table 5.9: Traceability Matrices Merges technique top-40 results.

similarity scores down.

5.5.2 Traceability Matrices Merge

Table 5.9 displays the result of the three strategies in both scenarios (threshold value 0.0 and 0.9) and top-40 cut.

The first strategy, the one that select the traces recovered by at least one of the traceability techniques (in other words, this strategy accept as a trace any recovered trace), reached a total of 93.58% recall rate in 0.0 threshold scenario. This means that the baseline study was able to recover 93.58% of the true positives traces, divided between the four techniques. However, the 5.16% precision result is not satisfactory. Low precision is related to the large number of false positives recovered by the traceability techniques, that end up also joining in the merged traceability matrix, since this is the least selective strategy. This strategy failed to recover only 33 true positives traces, as can be seen in Table 5.9.

The highest precision rate achieved by the traceability matrices merge technique was 29.17%, in the scenario that favors precision (0.9 threshold) and the most selective strategy (which also favors precision). Although this value is considered "acceptable", considering goodness table [17] (in 2.5.3), we noticed a very low recall rate of 1.36%. The traceability matrices merge technique failed to recover 507 true positives traces in this particular scenario.

The other two strategies are more selective, because they only select the traces that have been recovered by more than one technique, which reduces the number of both false positives

Trac. Technique	Improv. Technique	Precision	Recall	F-Score	Goodness
LSI	Text and Information Cleaning	20.11%	71.21%	47.21%	Acceptable
LSI	Spellchecker	20.16%	71.40%	47.34%	Acceptable
LSI	Techniques Combination	20.49%	72.57%	48.12%	Acceptable

Table 5.10: Goodness scale result after improvement techniques application: top-20 and 0.0 threshold.

and true positives. This can be seen in the Table, the more selective the strategy the greater the precision and the lower the recall.

5.6 Goodness Scale

In the baseline work [12] none of the traceability techniques were able to reach the level of "Acceptable" on the Goodness Scale, in the 0.0 and 0.9 threshold scenarios. However, three improvement techniques evaluated in this research were able to make the LSI reach the "Acceptable" level in the top-20 with 0.0 threshold.

Text and information cleaning (5.1), spellchecker (5.2), and techniques combination (5.4) increased the precision of the LSI technique to more than 20%, and precision was the only metric that had yet to reach the Acceptable Goodness Scale. The Goodness Scale result is displayed in Table 5.10.

5.7 Hypotheses Discussion

5.7.1 Hypothesis 1

Hypothesis: *the excessive textual information of a bug report contained in queries can impair the information retrieval process. By reducing this noise, we can achieve a better result.*

Contrary to what we imagined, removing the bug report fields with little relevant textual information does not influence the information retrieval process in all cases. In view of the similarity of results with the spellchecker technique, we believe that the punctuation removal process has more weight, in most cases, since punctuation removal is a common process between the two improvement techniques.

5.7.2 Hypothesis 2

Hypothesis: *bug reports are written by humans, so it is possible for typos to occur, which can impair the traceability recovery process.*

When applying the spellchecker technique *on the specific dataset used on our work*, we realized that there were not many typing errors, making the spelling correction process have zero impact on the automatic traceability recovery process. Most of the words classified as wrong by the spellchecker were, in this particular set of bug reports, technical terms, such as “mozilla”, “firefox”, “sqlite”, and “google”, among others.

The results shown in Section 5.2 are attributed to the textual cleaning process carried out before applying the spellchecker.

5.7.3 Hypothesis 3

Hypothesis: *some fields in the bug report may be more relevant to the traceability recovery process, so if we increase the weight of these fields in the creation of queries we can improve the results.*

According to the results shown in Section 5.3, between the title and description bug report fields, the title is the best field to increase weight, and it was the only one that showed a positive result. Because it is generally shorter than the description, the title may contain relevant terms with less textual noise.

By duplicating the description, which usually contains many more words without relevance to traceability, the importance of the relevant terms ends up being diluted in the query.

5.7.4 Hypothesis 4

Hypothesis: *if the traceability techniques used in the baseline work show different behaviors when recovering the traces, it is possible to join their effort.*

The similarity matrices merge technique is not able to porperly join the effort of the four traceability techniques in the 0.0 and 0.9 threshold scenarios. Even with LSI having the greatest weight in the calculation of the new similarity scores. We noticed that the negative influence of the other traceability techniques cause the similarity scores not to reach the values necessary for the traces recovery, resulting in a worse result until the LSI itself.

As seen in Subsection 5.5.2, the traceability matrices merge technique was able to join the effort of each traceability technique, achieving a 93% of recall, failing to recover only 33 true positives traces. However, in the baseline work [12], LSI alone had 88.33% recall in the top-40 with threshold 0.0, the individual efforts of the other three traceability techniques add up to only 5.25% of the recovered true positives traces, contributing a lot to the increase of false positives and negatives. Therefore, the traceability matrices merge was not able to overcome the flaws of the traceability techniques, recovering several false positives traces and continuing with the low result in terms of precision.

5.8 Threats to Validity

There are several threats to validity in this study. Although we explored dataset from a real and open-source project, we use only a single source of artifacts. So, the results obtained by this study cannot be generalized, because it was applied only on one system.

There is also a risk of overfitting, considering that we do not propose generic improvement techniques. The proposed techniques were developed with a focus on specifically improving the baseline study, which has a well-defined context, the Mozilla Firefox open source project. Finding this type of data (artifacts) is not an easy task, even more freely available on the internet and well structured as the artifacts used in this research.

Additionally, since we provide a tooling apparatus to automate many of the steps in the experiment, there is a risk that it contains some implementation errors not detected in the scripts used in the empirical studies.

As we use the same oracle traceability matrix as the baseline work, we have this oracle as a external threat to validity. The creation of the oracle traceability matrix had no participation of any member of the Mozilla's development and testing teams, so it is possible that this oracle traceability matrix contains some wrongly classified trace links.

The selection criteria applied to the set of bug reports is another threat to the validity, mainly because it limits the generalization of the results. Many times the bug reports are not well written as the selected ones, and we have seen that the quality of the bug description has an impact over the automatic traceability recovery task. Although, in order to draw solid conclusions using such real software artifacts, we needed to select the best quality ones.

Chapter 6

Related Work

Several studies analyze the effectiveness of using IR and ML techniques for traceability recovery between different software artifacts.

Gadelha [12] proposes and evaluates a study that uses three IR techniques (LSI, LDA, and BM25) and a DL algorithm (Word Vector) to recovery traceability traces between bug reports and test cases, and we use it as a baseline work. The chosen dataset was the artifacts of the Mozilla Firefox, because it is a open-source project and its artifacts are freely available online. Altogether 93 bug reports and 195 test cases were selected to be used in the study, the same dataset used in our proposal. Gadelha makes a detailed analysis in two different scenarios: (i) threshold value 0.0 and (ii) threshold value 0.9. As we chose to compare the results of our improvement techniques with the base work, we also chose the same scenarios. The threshold value 0.0 scenario was the one with the best general result in Gadelhas's work, reaching the F_2-Score values of 35.2% for LSI, 19.1% for LDA, 20.8% for BM25, and 7.9% for Word Vector. Our work has managed to slightly overcome these results, with the combination of the best improvement techniques, we have the following result, in the same scenario, in terms of F_2-Score: 38.3% for LSI, 20.8% for LDA, 22.3% for BM25, and 11.9% for Word Vector

Merten *et al.* [26] evaluate how effective IR is in tracking system artifacts. Altogether five IR techniques were applied: VSM, LSI, BM25, BM25+, and BM25L. Their study analyzes the traceability recovery between bug reports, to classifying them as *duplicates* or not, in four different open-source projects. The employed IR techniques were evaluated with and without text processing, and assigning different weights for title, description, comments, and code excerpts in bug reports. They also consider precision, recall, goodness scale, and two versions of F-Score: F_1-Score and F_2-Score. Despite the differences between ours and their study, the observed results are similar: all techniques performed poorly, with LSI performing the best. Increase the title weight also achieved a result improvement, same as in our work. Merten *et al.* also explore in their research some of the reasons why bug reports can be difficult to track, such as the presence of noise in the text and repetitive information.

Nilam Kaushit *et al.* [23] uses IR techniques (LSI and LDA) to recover the traceability between nine solved bug reports to more than 13,000 test cases from a private software. Two scenarios were evaluated, differing from each other by the grouping of test cases and the adopted relevance criterion. As evaluation metrics they also apply precision, recall, and F-Score. The only explored threshold value is 0.7, this threshold value was chosen because is a commonly used constant threshold.The top values are 2, 5, and 10. Another difference in relation to our work was in the creation of the oracle, Nilam Kaushit *et al.* had the help of a tester for the creation of the oracle, while the oracle used by us was that created in the baseline work, created by volunteers and a specialist (none of them belonging to the Mozilla Firefox team). As well as in our research, Nilam Kaushit *et al.* also observed better results in the IR technique LSI, reaching the best result of 45% in terms of F-Score. Our baseline work [12] evaluates two other techniques (BM25 and Word Vector), and our proposal adopt some improvement approaches. Despite using the same kind of artifacts (bug reports and test cases), our work uses a more significant amount of bug reports (altogether 93), which allows a more in-depth analysis. Regarding the test cases, they use a much larger amount than ours, with a total of 13,380 test cases. Additionally, our approach is applied on freely available online datasets from an open-source project, while the one used by Nilam Kaushit *et al.* on their research is private and anonymous.

Jin Guo *et al.* [16] proposes a tracing network architecture, employing Word Embedding and Recurrent Neural Network (RNN) models to generate trace links among several artifacts, such as requirements, design, test cases, and source code, by incorporating requirements semantics and domain knowledge. The training dataset is a corpus of clean text extracted from documents and software artifacts, including a Wikipedia dump containing about 19.92GB of text. To evaluate the tracing network, they recover links between Software Subsystem Requirements and Subsystem Design Descriptions. As a result, this approach outperforms state-of-the-art tracking methods, including the VSM and LSI techniques. Our approach, in turn, focuses on test cases and bug reports only. We evaluate some improvement techniques on a deep learning algorithm (Word Vector) applied by the baseline work, and our results are as well unsatisfactory. It is worth noting the great difference, between our work and theirs, in the amount of data used in training the DL algorithms.

Chapter 7

Conclusions

In this dissertation, we propose and evaluate a set of improvement techniques applied on a baseline work, an approach that uses IR and DL techniques to automatically recover traceability between bug reports and test cases on a dataset composed by Firefox system's artifacts. The employed improvement techniques are: (i) text and information cleaning; (ii) spellchecker; (iii) bug report fields weighting; (iv) merging the similarity matrices; and (v) merging the traceability matrices.

We evaluated those improvement techniques by measuring and comparing precision, recall, and f-measure, with the results reached by the baseline work. We observed a slight increase in precision and recall rates for all employed traceability recovery techniques (LSI, LDA, BM25, and Word Vector) with the application of the cleaning, spellchecker and the title duplication techniques when applied in isolation as when utilized together. One of the traceability matrices merging strategies achieved a recall value of 93%.

The reached results are encouraging and show that even the more basic NLP techniques are essential in the range of sophisticated techniques that address textual artifacts traceability. Additionally, merging traceability matrices reveal that exploring and combining different techniques adopted during matrix building is promising.

We recommend the application of the improvement techniques. Although the results were not as significant, the application of the improvement techniques evaluated in this research is not costly, and none of them modified the architecture of the approach in which they were applied.

A critical discussion is how impacting the bug report writing is for the traceability. The quality of bug reports is an explored study topic. From our study, we can point out factors that can negatively impact their automatic traceability recovery, such as (i) descriptions with just URL links or screenshot pictures; (ii) the misuse of keywords fields; (iii) titles and descriptions containing few textual information as well as different terms from those used in test cases. Even though the bug reports used in our research were well structured, with well-defined information fields, they were not immune to these problems. The lack of a well-defined structure can negatively affect the quality of bug reports, which further impacts the result of traceability techniques.

As we have not tested the approach in other projects, we have no idea of its generality. Both the bug reports and test cases of the Mozilla Firefox project have a well-defined structure, including the separation of test cases by system feature. We cannot guarantee that all projects will have an equal or similar structure, nor that the approach will behave similarly. The use of the approach described in this research in other projects is still an open topic.

7.1 Limitations

This research has some limitations that we must mention. As it uses a baseline work, we inherited some of its limitations, such as the traceability matrix used as an oracle, for instance. Since the process of creating the oracle involved some volunteers, with no Mozilla team representative, it is not highly reliable. To mitigate this problem, the oracle used by the baseline work (and consequently by us) was the one that contained the intersection between the answers of the specialist's oracles and the volunteers' oracles.

As the improvement techniques were designed and developed to improve the baseline work, the techniques did not have their generality explored. The evaluation of these techniques in other projects would give more evidence about the impact of each one of them in the process of recovering traceability.

We can also highlight the lack of a more detailed analysis on the statistical impact of applying the improvement techniques, such as the statistical significance. This more detailed analysis could give us more insights and robust evidences about the changes caused by the improvements techniques comparing to the baseline work.

7.2 Contributions

The main contributions of this dissertation are:

1. Application and evaluation of improvement techniques in a baseline case study that uses IR and DL techniques to recover traceability between bug reports and test cases from Mozilla Firefox;

2. A discussion about bug report in context of traceability.

7.3 Future Works

As future works we intend to apply and evaluate other improvement techniques, such as the adoption of a glossary or thesaurus to explore the semantic relationships and differences among the terms present in bug reports and test cases; explore other datasets as well as other textual artifacts, including a updated version of Firefox artifacts and others projects datasets, to evaluate the generality of the proposed techniques; research and evaluate other recovery strategies, such as other DL algorithms; and evaluate other similarity measures, such as Dice's coefficient and Jaccard index [25].

Another future work that is also valid is the creation of a new oracle traceability matrix, with a more reliable creation process that can present fewer human flaws.

Bibliography

[1] Paul Ammann and Jeff Offutt. *Introduction to software testing*. Cambridge University Press, 2016.

[2] Giuliano Antoniol, Kamel Ayari, Massimiliano Di Penta, Foutse Khomh, and Yann-Gaël Guéhéneuc. Is it a bug or an enhancement? a text-based approach to classify change requests. In *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds*, pages 304–318, 2008.

[3] Giuliano Antoniol, Gerardo Canfora, Gerardo Casazza, Andrea De Lucia, and Ettore Merlo. Recovering traceability links between code and documentation. *IEEE transactions on software engineering*, 28(10):970–983, 2002.

[4] Jairo Aponte and Andrian Marcus. Improving traceability link recovery methods through software artifact summarization. In *Proceedings of the 6th International Workshop on Traceability in Emerging Forms of Software Engineering*, pages 46–49, 2011.

[5] Hazeline U Asuncion, Arthur U Asuncion, and Richard N Taylor. Software traceability with topic modeling. In *2010 ACM/IEEE 32nd International Conference on Software Engineering*, volume 1, pages 95–104. IEEE, 2010.

[6] Daniel M Berry. Evaluation of tools for hairy requirements and software engineering tasks. In *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, pages 284–291. IEEE, 2017.

[7] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[8] Markus Borg, Per Runeson, and Anders Ardö. Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability. *Empirical Software Engineering*, 19(6):1565–1616, 2014.

[9] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[10] Mattia Fazzini, Martin Prammer, Marcelo d'Amorim, and Alessandro Orso. Automatically translating bug reports into test cases for mobile apps. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 141–152, 2018.

[11] William B Frakes and Ricardo Baeza-Yates. *Information retrieval: data structures and algorithms*. Prentice-Hall, Inc., 1992.

[12] Guilherme Monteiro Gadelha. An Aproach for Traceability Recovery between Bug Reports and Test Cases. *Dissertação (Mestrado em Ciência da Computação) - Universidade Federal de Campina Grande*, 2019.

[13] Yoav Goldberg and Omer Levy. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*, 2014.

[14] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. Deep learning. vol. 1, 2016.

[15] Orlena Gotel, Jane Cleland-Huang, Jane Huffman Hayes, Andrea Zisman, Alexander Egyed, Paul Grünbacher, Alex Dekhtyar, Giuliano Antoniol, Jonathan Maletic, and Patrick Mäder. Traceability fundamentals. In *Software and systems traceability*, pages 3–22. Springer, 2012.

[16] Jin Guo, Jinghui Cheng, and Jane Cleland-Huang. Semantically enhanced software traceability using deep learning techniques. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, pages 3–14. IEEE, 2017.

[17] Jane Huffman Hayes, Alex Dekhtyar, and Senthil Karthikeyan Sundaram. Improving after-the-fact tracing and mapping: Supporting software quality predictions. *IEEE software*, 22(6):30–37, 2005.

[18] Jane Huffman Hayes, Alex Dekhtyar, and Senthil Karthikeyan Sundaram. Advancing candidate link generation for requirements tracing: The study of methods. *IEEE Transactions on Software Engineering*, 32(1):4, 2006.

[19] Jeff Heaton. Ian goodfellow, yoshua bengio, and aaron courville: Deep learning, 2018.

[20] H Witten Ian and Frank Eibe. Data mining: Practical machine learning tools and techniques, 2005.

[21] Sascha Just, Rahul Premraj, and Thomas Zimmermann. Towards the next generation of bug tracking systems. In *2008 IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 82–85. IEEE, 2008.

[22] Vaishali Kalra and Rashmi Aggarwal. Importance of text data preprocessing & implementation in rapidminer. In *ICITKM*, pages 71–75, 2017.

[23] Nilam Kaushik, Ladan Tahvildari, and Mark Moore. Reconstructing traceability between bugs and test cases: An experimental study. In *2011 18th Working Conference on Reverse Engineering*, pages 411–414. IEEE, 2011.

[24] Tien-Duy B Le, Mario Linares-Vásquez, David Lo, and Denys Poshyvanyk. Rclinker: Automated linking of issue reports and commits leveraging rich contextual information. In *2015 IEEE 23rd International Conference on Program Comprehension*, pages 36–47. IEEE, 2015.

[25] Christopher D Manning, Hinrich Schütze, and Prabhakar Raghavan. *Introduction to information retrieval*. Cambridge university press, 2008.

[26] Thorsten Merten, Daniel Krämer, Bastian Mager, Paul Schell, Simone Bürsner, and Barbara Paech. Do information retrieval algorithms for automated traceability perform effectively on issue tracking system data? In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 45–62. Springer, 2016.

[27] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[28] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[29] Juan Ramos et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 133–142. New Jersey, USA, 2003.

[30] Raimundo Real and Juan M Vargas. The probabilistic basis of jaccard's index of similarity. *Systematic biology*, 45(3):380–385, 1996.

[31] Stephen Robertson and Hugo Zaragoza. *The probabilistic relevance framework: BM25 and beyond*. Now Publishers Inc, 2009.

[32] Stephen E Robertson. The probability ranking principle in ir. *Journal of documentation*, 33(4):294–304, 1977.

[33] Ripon K Saha, Matthew Lease, Sarfraz Khurshid, and Dewayne E Perry. Improving bug localization using structured information retrieval. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 345–355. IEEE, 2013.

[34] Shiwangi Singh, Rucha Gadgil, and Ayushi Chudgor. Automated testing of mobile applications using scripting technique: A study on appium. *International Journal of Current Engineering and Technology (IJCET)*, 4(5):3627–3630, 2014.

[35] Amit Singhal et al. Modern information retrieval: A brief overview. *IEEE Data Eng. Bull.*, 24(4):35–43, 2001.

[36] Ian Sommerville. Software engineering 9th edition. *ISBN-10*, 137035152:18, 2011.

[37] Jian Zhou, Hongyu Zhang, and David Lo. Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports. In *2012*

34th International Conference on Software Engineering (ICSE), pages 14–24. IEEE, 2012.