8-27-2018 2:30 PM

# Automatic Recall of Lessons Learned for Software Project Managers

Tamer Mohamed Abdellatif Mohamed, *The University of Western Ontario*

Supervisor: Capretz, Luiz F., *The University of Western Ontario*
Co-Supervisor: Ho, Danny, *The University of Western Ontario*
A thesis submitted in partial fulfillment of the requirements for the Doctor of Philosophy degree in Electrical and Computer Engineering
© Tamer Mohamed Abdellatif Mohamed 2018

## Recommended Citation
Mohamed, Tamer Mohamed Abdellatif, "Automatic Recall of Lessons Learned for Software Project Managers" (2018). *Electronic Thesis and Dissertation Repository*. 5615.
https://ir.lib.uwo.ca/etd/5615

# Abstract

Lessons learned (LL) records constitute a software organization's memory of successes and failures. LL are recorded within the organization repository for future reference to optimize planning, gain experience, and elevate market competitiveness. However, manually searching this repository is a daunting task, so it is often overlooked. This can lead to the repetition of previous mistakes and missing potential opportunities, which, in turn, can negatively affect the organization's profitability and competitiveness. In this thesis, we present a novel solution that provides an automatic process to recall relevant LL and to push them to project managers. This substantially reduces the amount of time and effort required to manually search the unstructured LL repositories, and therefore, it encourages the utilization of LL. In this study, we exploit existing project artifacts to build the LL search queries on-the-fly, in order to bypass the tedious manual search process. While most of the current LL recall studies rely on case-based reasoning, they have some limitations including the need to reformat the LL repository, which is impractical, and the need for tight user involvement. This makes us the first to employ information retrieval (IR) to address the LL recall. An empirical study has been conducted to build the automatic LL recall solution and evaluate its effectiveness. In our study, we employ three of the most popular IR models to construct a solution that considers multiple classifier configurations. In addition, we have extended this study by examining the impact of the hybridization of LL classifiers on the classifiers' performance. Furthermore, a real-world dataset of 212 LL records from 30 different software projects has been used for validation. Top-k and MAP, well-known accuracy metrics, have been used as well. The study results confirm the effectiveness of the automatic LL recall solution by a discerning accuracy of about 70%, which was increased to 74% in the case of hybridization. This eliminates the effort needed to manually search the LL repository, which positively encourages project managers to reuse the available LL knowledge – which in turn avoids old pitfalls and unleash hidden business opportunities.

## Keywords

Software lessons learned recall, software project management, knowledge management, information retrieval models, software analytics, systematic literature review.

## Acknowledgments

I am cordially thankful to my supervisors, Luiz Fernando Capretz and Danny Ho, for their advices, encouragement and time. I appreciate their constructive feedback and support through my PhD studies. They motivated me to refine my research and publish my work in top ranked conferences and reputable journals.

I dedicate this thesis to my wife Dina Kandil for her inspiration, patience and unlimited support.

In addition, I want to thank my family and my children Dorra and Youssef for their emotional support that helped me to keep up my passion throughout my research.

Also, I am grateful to my industrial partner for providing the dataset, which was necessary for the evaluation process of this work.

# Table of Contents

# List of Tables

# List of Figures

## List of Appendices

# List of Terms and Acronyms

| | |
|---|---|
| SA | Software Analytics: represents a branch of big data analytics. The term was initially coined by Zhang et al. in 2011 and it is concerned with the analysis of all software artifacts |
| SLR | Systematic Literature Review: type of literature review where a systematic method is followed to search, collect, extract and synthesize data to answer a set of defined review questions. |
| LL | Lessons Learned: the organization's memory of what went wrong (mistakes) or what went right (opportunities) in certain situations or events. |
| IR | Information Retrieval: refers to the process of finding a relevant document or information of interest within a collection of documents or artifacts. |
| VSM | Vector Space Model: is an algebraic information retrieval model. It represents the documents' corpus in a matrix format of terms versus documents |
| SVD | Singular Value Decomposition: a matrix decomposition method from linear algebra where a matrix is decomposed into three new matrices. |
| LSI | Latent Semantic Indexing: is an extension of the VSM model. It takes the context or topic into consideration instead of only matching the terms which can have different meanings. |
| LDA | Latent Dirichlet Allocation: is a generative probabilistic IR model. It considers the context of terms by eliciting the topics within the documents' corpus. |
| HSD | Tukey's Honestly Significant Difference: a statistical test with the ability to perform a comparison between different groups in one step. It can significantly differentiate between more than two groups based on the |

| | |
|---|---|
| | statistically significant difference between the groups' mean. |
| Top-K | An accuracy metric calculates the percentage of queries, which are executed by the IR model, in whose top k result records there is at least one relevant record. |
| MAP | Mean Average Precision: a popular accuracy metric from the IR literature. It is calculated as the average of the aggregated average precision of each individual query. |
| Borda Count | A rank-based hybridization technique. The total hybrid score of each retrieved item is calculated as the summation of the item ranks from each individual classifier retrieval list. |
| Score Addition | A hybridization technique relies on the item's weight. The total hybrid score of each retrieved item is calculated as the summation of the individual score of this item from each of the combined classifiers. |
| RI | Relative Performance Improvement: an improvement percentage which is calculated by comparing the result of the hybrid classifier to that of the classifier with the highest performance among the individual classifiers within the combination set. |

# Chapter 1

## 1 Introduction

Everyday situations teach us lessons and provide us with analogs. We rely on these lessons learned (LL) and analogs in dealing with similar problems and identifying opportunities. Accordingly, the decision making literature [1] tells us about the power of analogs. Analogs are experiences or knowledge from domains similar to that of the subject at hand and can work for different domains such as management, decision making, and entrepreneurship. They can support decision making by providing recommendations, improving contextual awareness, facilitating difficult tasks, clarifying problem definition, and fostering development of reasonable plans. Thus, managers can rely on analogs to facilitate their decision making. Also, entrepreneurs can rely on analogs to build their business models and to test their hypothesis [2].

The software engineering domain is no exception, regarding analogs, as learning from similar projects or previous experiences improves the success and quality of software projects. This was highlighted by Pfleeger [3] when she used an analogy from soccer. In a soccer game, the team does not just play the game and then forget about it. Instead, a good team comes together after the game to discuss and analyze their performance, what happened and what can be improved. They define the LL from their performance and analyze the game scenarios and circumstances. They make use of these LL to overcome mistakes, improve their performance and leverage good plays. It can be the same for software engineering. Project managers (PMs) can make use of their experiences and conduct post-action meetings in order to discuss, analyze and report LL.

Analogs can come from accumulated personal or organizational knowledge or experience. For any organization, local LL can provide a precious and reliable source of applicable analogs since LL are generated from the same organization’s culture and work environment. This makes LL more convenient and fitting for the organization’s current and future projects and customers. LL records can contain information regarding either a positive experience, such as a business opportunity, or a negative experience such as a mistake or a problem. Both positive and negative experiences are valuable since they can either highlight an opportunity for leveraging profit or eliminate an anticipated loss, respectively. That said, reporting and safely storing such historical knowledge, or LL, is encouraged by different standards from reputable management institutions such as the Project Management Body of Knowledge (PMBOK) by the Project Management Institute (PMI) [4] and the Capability Maturity Model Integration (CMMI) [5]. Both standards emphasize the importance of conducting closing reviews and retrospective meetings to record and analyze LL from previous projects for future reference. Because of the high value placed on LL, the National Aeronautics and Space Administration (NASA) has assigned an effort and budget to improve its LL reporting process and launch its LL portal [6].

It is worth mentioning that it is not the existence of data itself— the LL records in our case— that is valuable, but rather the awareness of the appropriate and relevant records and the ability to manipulate these LL records to deal with needs. Keeping this in mind, we should ask what happens when PMs overlook LL records, ignore them, or do not even know of the existence of relevant LL records.

*After the attack on Pearl Harbor in 1941, the Japanese commander, Mitsue Fuchido, was surprised by its success. He asked, "Had these Americans never heard of Port Arthus?" that event, which was famous in Japan, had preceded the Russo-Japanese war of 1904. The Japanese tactic was to destroy the Russian Pacific fleet at anchor at Port Arthur in a surprise attack.*

*Wohlstetter 1962 [1]*

Overlooking or missing relevant LL records can lead not only to missing available opportunities and business success, but can also cause a catastrophic loss of profit due to the repetition of known previous mistakes which could be easily avoided by just knowing about them. In this thesis, we have worked on the enhancement of the retrieval of relevant LL records to make them available to PMs. Our goal has been to facilitate access to these LL records in the organization's repository, and therefore minimize or eliminate the problem of overlooking LL. Accordingly, our solution, automatically recalls relevant LL records and provides them to PMs to avoid the intensive effort needed to manually search for them. In order to build this solution, we have employed information retrieval (IR) techniques, for the first time within the LL recall context. In addition, we have manipulated two of the existing project management artifacts in order to construct search queries automatically instead of searching manually by PMs.

## 1.1 Motivation

Our software analytics (SA) systematic literature review (SLR), described in Chapter 2, has shown that most of the available SA studies address issues that serve developers (about 90% of the studies), while a few studies target other stakeholders such as management and quality assurance teams (See Section 2.2.2). Also, the SLR results have shown that the majority of studies analyzed only a few of the artifacts accompanying the software development lifecycle, with many of the studies analyzing only source code. The focus of this thesis is twofold. First, we decided to focus the research on providing a solution to serve stakeholders other than developers. Second, we aimed to exploit the software management artifacts that were not heavily analyzed in previous studies in the software engineering literature. By concentrating the research point on these two axes, we have worked towards closing the research gap, as revealed by the SLR, and tackle a genuine research challenge. With this motivation in mind, we decided to define the general research topic as serving software PMs. Also, we have focused our thinking on a solution that would have a high impact and promote knowledge. In order to narrow down the research topic and define specific research areas, we started by surveying the LL literature. The survey indicated that there are a small number of existing LL studies. In addition, most of these studies were focused on the implementation of a standalone LL system. Beyond this, the LL literature survey has clarified that there is a problem in the dissemination or recall of LL, and that this can be attributed to the intensive task of manually searching the LL repository for relevant LL. Finally, we have noticed that most of the available LL recall studies employed case-based reasoning techniques, which have some practical limitations (See Section 3.2). These observations led us to defining the research gap which we have worked to close in the solution provided in this thesis work. As will be described in detail, we have provided a solution to improve LL recall using IR techniques. The solution serves software PMs, and it can also be generalized to serve PMs in other domains, by providing them with LL records relevant to the projects at hand.

As described, the recall of LL is not currently being optimized due to the challenge of manually searching for relevant LL. This can lead to overlooking this valuable knowledge and losing precious opportunities. As a result, in this thesis, we aim to address this issue in order to facilitate LL recall, which in turn improves the benefit to stakeholders from existing LL knowledge. With our motivation clarified, in the next section, we will describe the problem we have addressed.

## 1.2 Problem Statement and Research Questions

Every aspect of a software PM's job is about predicting the future and anticipating problems and outcomes. Information technology organizations and project management offices (PMO) usually have LL repository systems. This LL repository can be viewed as the organization’s memory, as it contains a wealth of historical experiences. These LL records can provide valuable analogs for PMs which can facilitate the decision-making process and make decisions more accurate, rational and reasonable.

The problem is that these repositories are rarely reviewed or referenced by PMs during the project lifecycle, especially the project initiation phase. This can lead to the repetition of previous mistakes. It also means that opportunities for benefiting from previous success stories are missed. Discarding lessons learned repositories can be the result of the intensive nature of manually searching for relevant lessons learned by PMs, or other reasons such as time limitations or lack of awareness of the presence of relevant LL.

In this work, we provided a solution to address this problem by improving the recall of LL records. As aforementioned, we employed IR techniques to implement this solution. That said, we defined two main research questions in order to implement and validate the proposed solution as follows (See Section 4.2):

*RQ1: Can we automatically, rather than manually, recall and push relevant LL to PMs using IR-based LL classifiers?*

*RQ2: Can project artifacts be used to construct on-the-fly queries to recall LL records relevant to the software project at hand?*

As shown in these two research questions, we sought a solution to the issue of the intensive effort required to manually search the LL repository for relevant records. Our other aim in this solution was to avoid the existing limitations of previous studies, which employed case-based reasoning (See Section 4.3), by employing IR techniques for the first time in the LL recall context as per our knowledge, to construct an LL classifier to retrieve the LL relevant records to the project at hand. In the second research question, we intended to examine whether we could exploit any of the existing project management artifacts to automatically construct queries to search for relevant LL records. By constructing the queries automatically, we avoided the need for the manual involvement of PMs to define the queries, which could lead to overlooking LL records due to time limitations and the effort required. In order to answer the research questions and validate the solution, we conducted an empirical study. In this study, we employed a real dataset from an industrial partner. The dataset considered contained records for both LL and project management artifacts (See Section 5.1). The study results confirmed the validity of the solution in recalling relevant LL efficiently (See Sections 5.3 and 5.4).

In addition, we considered multiple IR techniques and classifier configurations. Since the LL classifier's configuration could affect its performance, we examined the impact of configuration by defining the following research question:

*RQ3: Do the configurations of LL classifiers have an impact on the performance results?*

In order to answer this research question, we conducted a statistical test on the results considering different classifier parameter configurations (See Section 4.4.2).

After recording the LL recall results, we performed one more step by examining the impact of the hybridization of multiple classifiers on the accuracy of the LL retrieved. Therefore, we considered two of the hybridization techniques from the literature (See Section 6.2) and sought an answer to the last research question:

*RQ4: Can hybridization improve the LL recall accuracy?*

We answered this question by first constructing multiple hybrid classifiers using combinations of the individual classifiers from our empirical study. Second, we compared the hybrid classifiers performance to that of the individual classifiers and calculated the relevant improvement. The results showed improvement in many cases, which encourages the consideration of hybridization for future studies in the LL recall context (See Section 6.4).

## 1.3 Research Contributions

The contribution of our research includes the comprehensive SLR which we conducted in order to come up with the SA state-of-the-art. The SLR results contribute to the software engineering community by providing valuable insights and defining research gaps. In addition, our research contributes by providing an LL automatic recall solution. Therefore, the main contributions of this thesis can be summarized as follows:

1. It provides both researchers and practitioners with a vision of the SA state-of-the-art to support them, in focusing their research on the research trends and important domains (See Section 2.2.2.2). This survey clarifies the research gaps and research opportunities within the SA domain. It can also facilitate the selection of future research topics and projects for interested researchers (See Sections 2.2.2 and 2.4).

2. It is first to tackle the LL recall problem within the software engineering domain and to provide an LL recall solution.
3. It is the first time, as per our knowledge, that IR models have been used to build an LL recall classifier (See Section 4.3). It bypasses the limitations of existing studies, which require the close involvement of users and reformation of the LL repository. This is impractical (See Section 3.2).
4. It exploits two of the existing software project management artifacts to create the search query on-the-fly. This relieves PMs from the burden of manually searching and facilitates the recall of relevant LL records (See Section 4.3.2). It also examines the impact of the classifier configurations on the LL retrieval accuracy.
5. It extends the main case study by examining the impact of hybridization on the accuracy of LL recall. The results showed a relative improvement in the hybrid classifier versus the individual classifiers in many cases. This can encourage the consideration of hybridization in future LL recall studies.

Table 1-1 summaries the mapping of the research questions to their corresponding research contributions and the chapters where the RQs are addressed.

**Table 1-1 Research Questions to Research Contributions Mapping**

| Research Question | Corresponding Contribution | Chapter |
|---|---|---|
| **SLR questions and results** | Contributions 1 | Chapter 2 |
| **RQ1** | Contributions 2 and 3 | Chapters 3, 4 and 5 |
| **RQ2** | Contribution 4 | Chapters 4 and 5 |
| **RQ3** | Contribution 4 | Chapters 4 and 5 |
| **RQ4** | Contribution 5 | Chapter 6 |

## 1.4 Thesis Structure

This thesis is organized as follows: Chapter 2 presents the methodology and results of the software analytics systematic literature review. Chapter 3 describes the related terms and concepts used in this thesis, including the LL and IR description, followed by an LL recall literature survey. In Chapter 4, we introduce the proposed LL recall solution, research questions and research methodology. The case study and validation of the proposed solution are illustrated in Chapter 5. In Chapter 6, we extend our main study by examining the classifiers hybridization impact on performance. Finally, Chapter 7 summarizes this thesis work and proposes future work.

# Chapter 2

# 2 Software Analytics Systematic Literature Review

In this chapter, we will describe the software analytics (SA) concept and we will present the state-of-the-art of the current SA research. Moreover, we will explain the protocol defined to conduct the extensive SA systematic literature review (SLR), and demonstrate the main findings and insights.[1]

## 2.1 Software Analytics

SA represents a branch of big data analytics. The SA term was initially coined by Zhang et al. in 2011 [7]. SA is concerned with the analysis of all software artifacts, not only source code. Its importance comes from the need to extract insights and facts from available software artifacts to support and facilitate decision making. Artifacts are available from all software development life cycle steps, beginning with the proposal and project initiation phases and ending with the project closure and customer satisfaction surveys. The dynamic nature of the software industry is associated with decision-making needs through all software business tiers. These tiers vary from the senior management board, setting the enterprise vision and portfolio management, going through project management planning and implementation by software developers. As emphasized by some experts [8][9][10][11] in the SA domain, all of the stakeholders involved deserve to be supported with decision-making tools in order to facilitate the decision-making process. SA can play the role of tool provider by offering suitable and supportive insights and facts to software industry stakeholders to make their decision making easier, faster, more precise, and more certain. The main difference between SA and direct software analysis is that rather than just providing straightforward insights extraction, SA performs

[1] Part of this chapter was published in the IEEE International Conference on Software Engineering Workshop on BIG Data Software Engineering.

**Tamer Mohamed Abdellatif**, Luiz Fernando Capretz, and Danny Ho, "Software Analytics to Software Practice: A Systematic Literature Review," in *Proceedings of the 37th IEEE International Conference on Software Engineering (ICSE) Workshop on BIG Data Software Engineering (BIGSE),* 2015, Florence, Italy, pp. 30–36.

additional advanced steps. As clarified by Hassan [8], SA provides both visualization and useful interpretation of insights in order to facilitate decision making.

## 2.2 Systematic Literature Review

In this section, we describe in detail the protocol employed to conduct the SLR regarding the SA research field. We discuss the results of this SLR, which represent the state-of-the-art of SA within the software engineering literature.

Since SA is currently a promising topic of broad interest, we have conducted an SLR to identify gaps in knowledge and open research areas in SA. Because many researchers are still confused about the true potential of SA, we had to filter out the available research papers to obtain the most SA-relevant work for our review.

In the SLR, we followed Kitchenham's [12] approach for a software engineering literature review. As a result, we started with the planning phase, in which we developed the review protocol.

In the following subsections, we start by describing the defined protocol of the SLR. Then, we demonstrate the results and findings of our study.

### 2.2.1 Systematic Literature Review Protocol

We conducted the review in six stages: defining questions, designing the search strategy, selecting the studies to use, assessing the quality, extracting data, and synthesizing data. The following subsections describe in detail each of the six stages considered.

#### 2.2.1.1 Defining Questions for Systematic Review

In this SLR, we answered four main questions. We kept in mind, while defining these questions (Q's), the two main targets of defining research gaps and defining opportunities within the SA field. The questions, for our SLR, are as follows:

***Q1: Which software practitioners does the available SA research target?***

Q1 aims to identify the stakeholders who will benefit from the available SA studies. It also aims to assess whether SA studies target different levels of stakeholders or only focus on the software development team in order to draw the attention of the SA research community to improve the research plan.

***Q2: Which domains are covered by SA studies?***

Q2 tries to highlight the scope of the available SA studies. The target domains, such as software maintainability and incident management, will be determined. Practitioners can interpret this information from two points of view. The first point of view is to know SA hot topics and consider them for their research plan, while the other view is to analyze any research gap and take the lead to consider it as an original research point.

***Q3: Which software artifacts are extracted?***

The main difference between SA and direct software analysis is making use of all of the available artifacts in order to come up with insights for strong decision making. Therefore, Q3 aims to verify that this idea is clear for the current research community.

***Q4: If different artifacts are used, are they linked together?***

Q4 tries to evaluate whether each study satisfies SA's main focus of linking different software artifacts. This linkage aims to come up with more advanced insights, unlike direct software analysis and metrics where researchers use each artifact separately without linkage to other artifacts.

#### 2.2.1.2 Search Strategy

Designing the search strategy is crucial and has a direct impact on the SLR results and concluding insights. The search strategy stage is two-pronged and includes defining the search terms and determining which software engineering literature libraries will be considered. In the following two subsections, we demonstrate our decisions regarding these two steps. Also, the rationale behind our decisions is illustrated.

#### 2.2.1.2.1 Search Terms Definition

To guarantee that the review would be closely relevant to SA, we tried to limit the search to the most SA relevant search term. So, we started with the term "software analytics", then we went through the following four steps in order to come up with the final search term that would be considered:

1. Extracting the major distinct terms from the questions.

We ran a small prototype using the original "software analytics" search term. In this prototype, we analyzed about 20 of the search result papers. From this analysis, we noticed the usage of different spellings of the original "software analytics" term.

2. Using different spellings of the terms.

As described in step 1, different spellings of the main keywords from the original search term were noticed. These spellings included the use of both singular and plural forms of the keyword "analytic". Also, the term "development" was sometimes associated with the original search term.

3. Updating the search term with keywords from relevant papers.

We considered the different keywords we had noticed in updating and considering different parts within the final search term.

4. Using the main alternatives and adding the "OR" operator in order to get the maximum amount of directly relevant literature.

These steps yielded the following search term:

***"Software analytics" OR "Software analytic" OR "Software development analytics" OR "Software development analytic"***

It is worth to mention that we did not consider the popular term of "software analysis" as we noticed that it led to a lot of studies, i.e., papers, which deal with traditional source code analysis studies. Since in our SLR we focused on the most current and the state-of-the-art studies targeting more than one software artifact and stakeholders other than developers, we preferred to exclude this term, i.e., "software analysis," from the search term. Instead, we analyzed most of the studies which use this term and are referenced by other studies within the primary studies list. We considered any of these studies that satisfied the filtration criteria as described later in this chapter.

##### 2.2.1.2.2 Literature Libraries Resources

In this SLR, we included two of the most popular electronic databases in order to search for the primary review studies, namely *IEEE – Xplore* and *ACM Digital Library*. The search term was constructed using the advanced search features provided by each of these two databases. The search covered metadata in the case of *IEEE – Xplore,* and both metadata and body (content) of literatures in the case of *ACM Digital Library.*

Our search included the period January 2000 to December 2014. Since the SA concept was initially introduced by Zhang et al. in 2011 [7], we expected that relevant literature would be found from 2011 and forward. However, we made our search timeframe wider in order to guarantee gathering all possible relevant papers.

#### 2.2.1.3 Study Selection

In order to eliminate any irrelevant papers which would not add any significant information, we conducted the following two filtration phases:

• Filtration phase 1: both inclusion and exclusion criteria (as defined in the next subsection) were defined and applied to the unique candidate papers to eliminate any irrelevant papers so that only relevant papers with useful information would result from this phase.

• Filtration phase 2: the quality assessment criteria (as defined in the next subsection) were used to assess candidate papers that emerged from phase 1. The papers which satisfied the quality boundary were used in the data extraction stage.

#### 2.2.1.3.1 Inclusion and Exclusion Criteria

As mentioned above, carefully defining the inclusion and exclusion criteria is crucial in leveraging the chance of including only relevant studies from the search results. For this reason, we defined the inclusion and exclusion criteria as follows:

**Inclusion Criteria**

The studies that satisfied the following four criteria passed the first filtration phase and were considered for the quality assessment step or second filtration phase. The inclusion criteria are defined as follows:

• SA concepts were applied to extract insights from software project artifacts.

This criterion was defined to guarantee the alignment between the considered study and the SA definition.

• Research was relevant to software project lifecycle phases.

Again, this criterion was defined to guarantee the fulfilling of the SA definition.

• Research was directly related to the software industry and stakeholders.

This criterion originated from the SLR prototype, as we noticed the usage of the SA term within studies from domains other than software engineering which just refer to the SA term.

• For duplicate publications of the same study, the newest and most complete one was selected. This is recorded for only one study whose related work appeared in two conferences.

**Exclusion Criteria**

When any of the following criteria applied to a study, we excluded it from the list of papers to consider. The exclusion criteria defined for this study were as follows:

• Studies that were irrelevant to SA.

This occurs due to the misuse of the term “software analytics” for describing traditional data mining, machine learning, or statistical work.

• Studies that were irrelevant to software projects.

This includes studies targeting other domains such as the automotive industry that misuses the term “software analytics” to refer to general “data analytics.”

• Studies that are relevant to generic data analytics and are not directly relevant to SA or software artifacts.

##### 2.2.1.3.2 Review Quality Assessment

This step was important to ensure the accuracy of data extraction from the studies reviewed and in order to be confident about the results and conclusions. Also, as previously mentioned, this step was considered to be the second filtration step in order to come up with the final primary studies to consider while answering the questions. We defined the following quality assessment criteria:

QA1: The study contribution is clearly stated.

QA2: Software artifacts that are used are clearly explained.

QA3: SA characteristics are clear and different from those of direct statistics where advanced insights are provided.

QA4: The results and application(s) are described in detail.

Each of the quality assessment criteria has only three optional answers: “Yes” = 1, “Partly” = 0.5 and “No” =0. For each study, the quality score is the sum of the scores of each quality assessment point, and the overall score is adjusted to a percentage scale. For this study, the quality assessment was used mainly as a selection criteria, as previously mentioned, based on the limitation that the papers considered were only those which had a quality score ≥ 50%.

It is worth mentioning that for the scoring process, we adopted the scoring method recommended by Kitchenham [12] in the case of having only one main researcher. So, the main researcher, the PhD candidate in this case, scored the studies based on his judgment and experience. Then, in this case, a review check was performed by the research team, the PhD supervisors and teammates. For this review check, the principle researcher went through the scores of the studies considered. In case of any concern or disagreement, a discussion meeting was scheduled between the researchers where a discussion took place until reaching an agreement regarding all scores of the studies considered. The review check process is a repetitive process and can be repeated until reaching a confident level of agreement among researchers regarding the assigned scores. For this study, the review discussion was finalized in one discussion meeting of about one hour. A summary of the scoring process is shown in Figure 2-1.

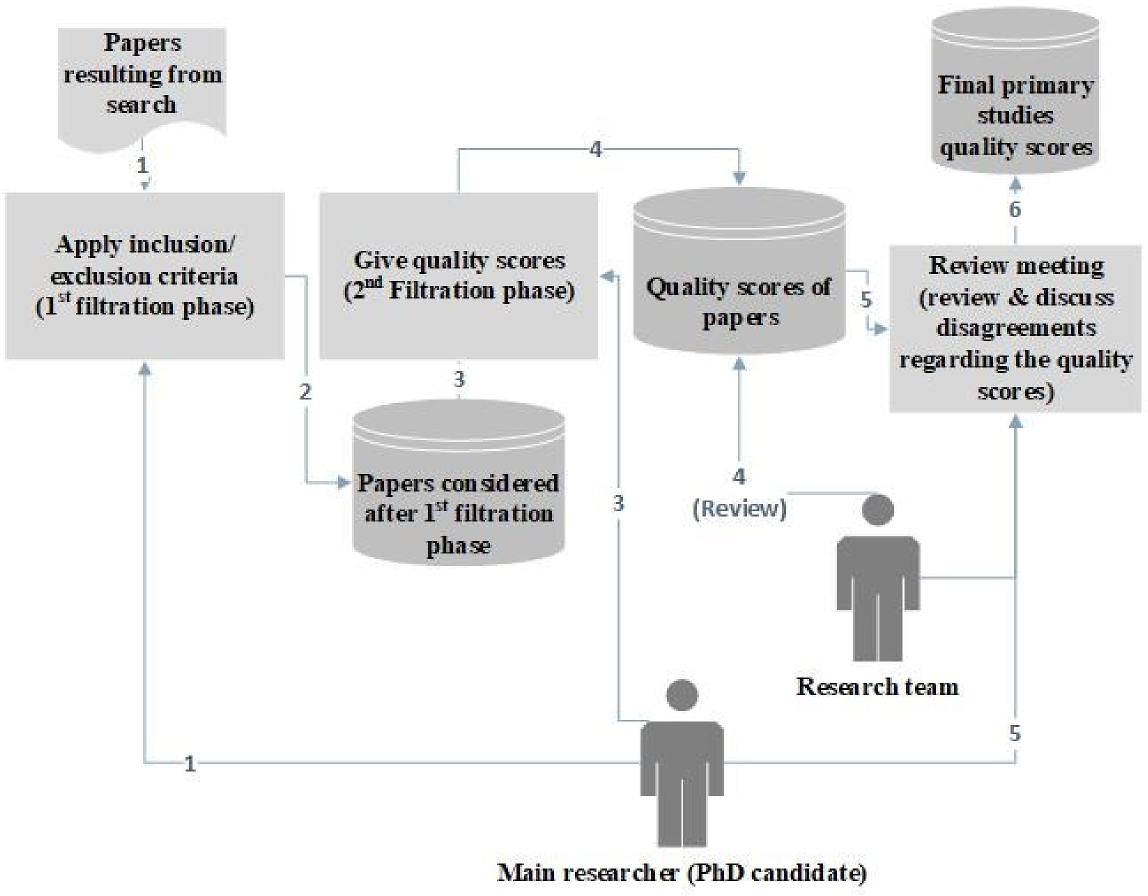
Papers resulting from search
1
Apply inclusion/ exclusion criteria (1st filtration phase)
2
Papers considered after 1st filtration phase
Give quality scores (2nd Filtration phase)
3
4
Quality scores of papers
5
Review meeting (review & discuss disagreements regarding the quality scores)
6
Final primary studies quality scores
4 (Review)
Research team
3
1
5
Main researcher (PhD candidate)

Figure 2-1 Quality Assessment Scoring Process

### 2.2.1.4 Data Extraction

The data extraction step is two- pronged. First, it involves defining the extracted pieces of information or the data which should be obtained from each study considered, in order to answer the questions. Second, these defined pieces of information for each of the studies considered need to be extracted and stored to prepare for the analysis and data synthesis stages as described in the next subsection.

We defined the data extraction card as shown in Figure 2-2. It is important to be very careful when defining fields or pieces of information to be included in the data extraction card, since all of the information required to answer all of the questions should be collected and made available for the analysis phase. This is crucial, because discovering missing information at the analysis stage will be very expensive and can lead to an intensive process of going through all of the studies that were considered to extract this missing information. For this reason, the data extraction card was carefully reviewed, and then a pilot study was executed on a small sample of the studies being considered in order to be confident that the questions could be answered using the information from the data extraction cards. Once we were confident of the completeness of the defined data extraction card, we ran the data extraction process on all of the studies considered.

Study id
Authors
Study title
Source
Year of publication
Q1: Beneficiary practitioners
Q2: Domain
Q3: Analyzed software artifacts
Q4: Different linked artifacts

**Figure 2-2 The Data Extraction Card**

### 2.2.1.5 Data Synthesis

In this stage, the extracted data was aggregated in order to answer the questions. For the questions answers representation, we used the narrative synthesis method. Accordingly, we used tables and charts to present the results.

After defining the protocol, the next step was to execute this protocol in order to come up with the primary studies, and then extract and record the needed information. After that, we applied data synthesis on the recorded data to come up with and report our insights and conclusions. Moreover, as a final step, we reported in detail the review limitations and provided our recommendations in the next section.

A summary of the protocol definition and the conduction processes for the SLR is shown in Figure 2-3.

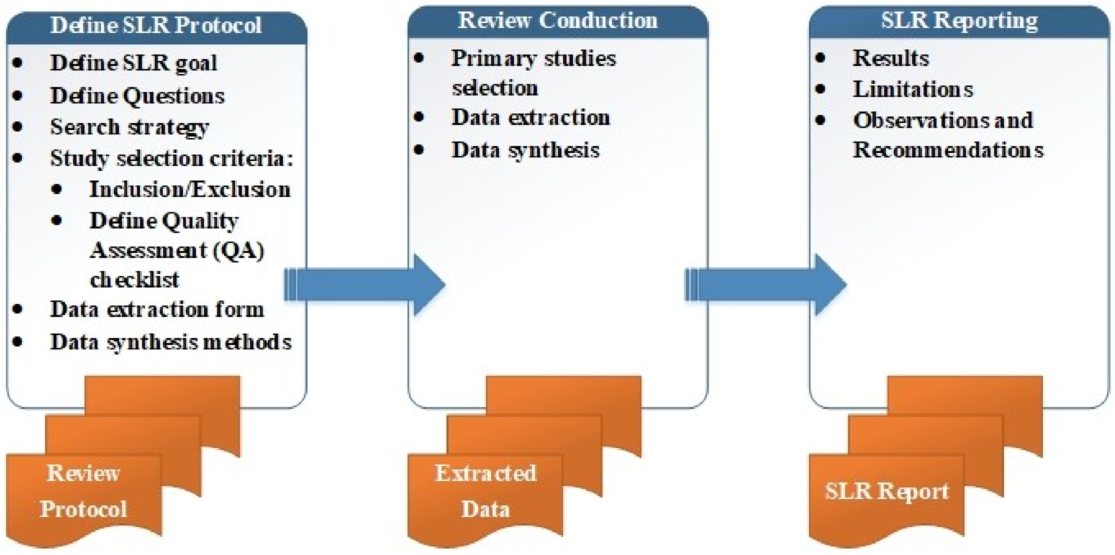

**Figure 2-3 A Summary of the SLR Process**

## 2.2.2 Systematic Literature Review Results

We followed the defined protocol (see Section 2.2.1), in order to execute the SLR. As a result, we started by searching the libraries we had decided on using the defined search terms. The search results contained 135 unique candidate papers (41 papers from IEEE Xplore, 102 from ACM Digital Library). There were 8 duplicate papers, for which we considered only one version.

The next step was to apply the two filtration steps. By applying both inclusion and exclusion criteria, the relevant papers totaled 41. After applying phase 2 of the filtration process, represented by the quality assessment stage, the relevant papers were narrowed down to 19; these papers were used for data extraction. The list of selected primary studies is shown in Table 2-1, and their correspondence quality scores are shown in Table 2-2. Also, the filtration process is summarized in Figure 2-4.

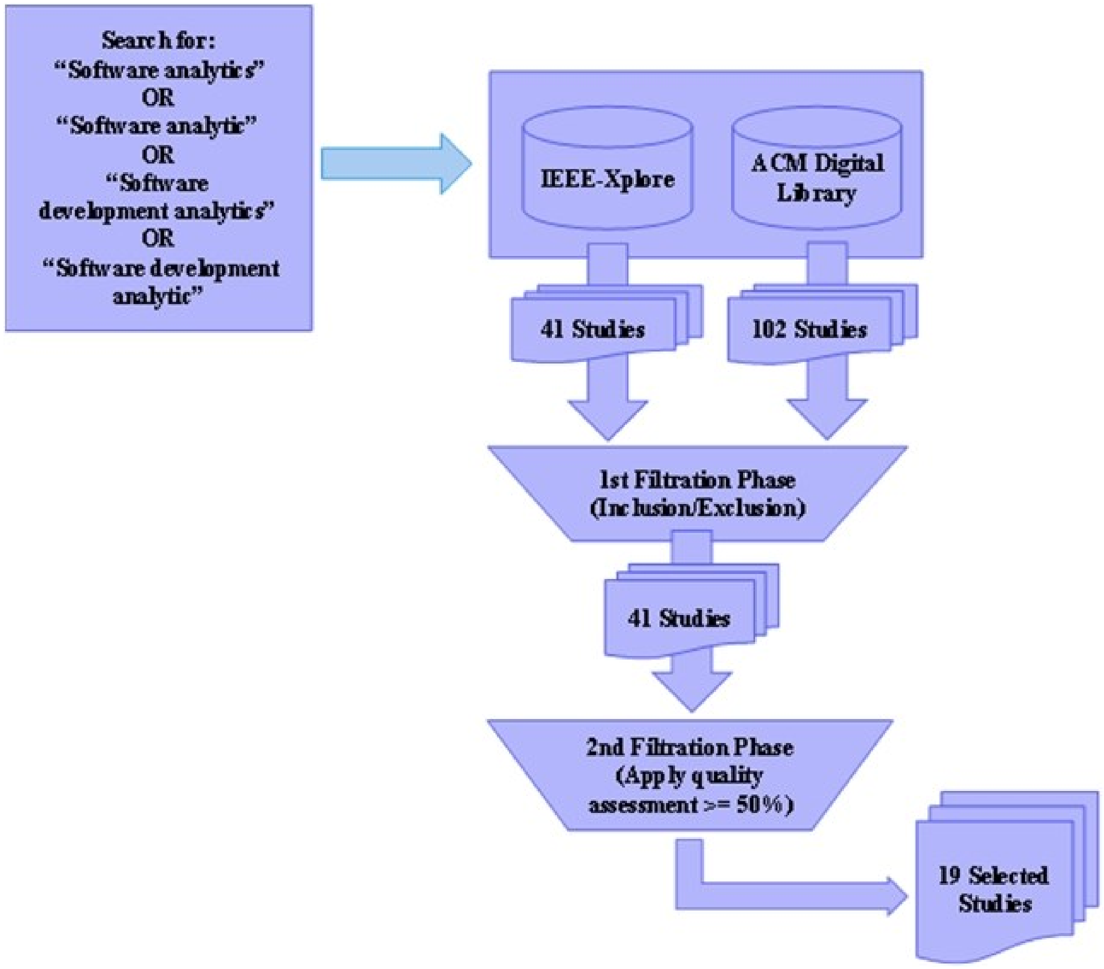
Search for: "Software analytics" OR "Software analytic" OR "Software development analytics" OR "Software development analytic"
IEEE-Xplore
ACM Digital Library
41 Studies
102 Studies
1st Filtration Phase (Inclusion/Exclusion)
41 Studies
2nd Filtration Phase (Apply quality assessment >= 50%)
19 Selected Studies

**Figure 2-4 Filtration Process**

**Table 2-1 Primary Studies Selected**

| ID | Authors | Addressed Questions | | | | Reference |
|---|---|---|---|---|---|---|
| S1 | M. van den Brand et al. | 1 | 2 | 3 | 4 | [13] |
| S2 | A. Gonzalez-Torres et al. | 1 | 2 | 3 | | [14] |
| S3 | E. Stroulia et al. | 1 | 2 | 3 | 4 | [15] |
| S4 | D. Reniers et al. | 1 | 2 | 3 | | [16] |
| S5 | R. Minelli and M. Lanza | 1 | 2 | 3 | 4 | [17] |
| S6 | J. Lou et al. | 1 | 2 | 3 | 4 | [18] |
| S7 | C. Klammer and J. Pichler | 1 | 2 | 3 | | [19] |
| S8 | T. Taipale et al. | 1 | 2 | 3 | 4 | [20] |
| S9 | O.Baysal et al. | 1 | 2 | 3 | | [21] |
| S10 | P. Johnson et al. | 1 | 2 | 3 | 4 | [22] |
| S11 | J. Czerwonka et al. | 1 | 2 | 3 | 4 | [23] |
| S12 | J. Gong and H. Zhang | 1 | 2 | 3 | 4 | [24] |
| S13 | A. Miranskyy et al. | 1 | 2 | 3 | 4 | [25] |
| S14 | R. Wu et al. | 1 | 2 | 3 | | [26] |
| S15 | S. Han et al. | 1 | 2 | 3 | | [27] |
| S16 | Y. Dubinsky et al. | 1 | 2 | 3 | | [28] |
| S17 | N. Chen et al. | 1 | 2 | 3 | | [29] |
| S18 | M. Mittal and A. Sureka | 1 | 2 | 3 | | [30] |
| S19 | G. Robles et al. | 1 | 2 | 3 | 4 | [31] |

By defining the primary studies to consider, we employed the defined data extraction card to extract the information needed to answer the questions and execute the data synthesis stage.

The dominant observation of this review was that there was not much relevant or well established research in the field of SA. This was clear from the number of papers considered (19) after applying both filtration phases, as explained earlier. The number of publications shown included all studies that were available and reviewed. Results showed that about 79% of the considered papers (15) were from conferences, while the remaining 21% (4) were from journals. Furthermore, almost all journal papers (3) were from *IEEE software* and were included in SA special edition published in 2013. These statistics emphasize the difficulty we faced in finding mature SA work for this review. As mentioned in the quality assessment section, we considered only the papers with a quality score ≥ 50% in order to guarantee including the most relevant studies. Most of the studies considered have a quality score ≥ 75% (15 out of 19 papers). Table 2-3 shows the quality score levels of all papers that passed the first filtration phase.

**Table 2-2 Primary Studies Quality Scores**

| **Study ID** | **QA1** | **QA2** | **QA3** | **QA4** | **Score** |
|---|---|---|---|---|---|
| S1 | 1 | 1 | 0 | 1 | 75% |
| S2 | 1 | 1 | 0 | 1 | 75% |
| S3 | 1 | 1 | 1 | 1 | 100% |
| S4 | 1 | 1 | 0 | 1 | 75% |
| S5 | 1 | 1 | 0.5 | 0.5 | 75% |
| S6 | 1 | 1 | 1 | 0.5 | 87.5% |
| S7 | 0.5 | 1 | 0 | 0.5 | 50% |
| S8 | 1 | 1 | 1 | 0.5 | 87.5% |
| S9 | 1 | 1 | 0 | 1 | 75% |
| S10 | 1 | 1 | 1 | 0.5 | 87.5% |
| S11 | 1 | 1 | 1 | 1 | 100% |
| S12 | 1 | 1 | 1 | 1 | 100% |
| S13 | 1 | 1 | 1 | 1 | 100% |
| S14 | 0.5 | 1 | 0 | 0.5 | 50% |
| S15 | 1 | 1 | 0 | 0.5 | 62.5% |
| S16 | 0.5 | 1 | 0 | 0.5 | 50% |
| S17 | 1 | 1 | 0 | 1 | 75% |
| S18 | 1 | 1 | 0.5 | 1 | 87.5% |
| S19 | 1 | 1 | 1 | 1 | 100% |

The distribution of the studies selected in each publication year is shown in Figure 2-5, which clearly indicates that SA studies became more active only in the last two years, 2013 and 2014.

In the following subsections, we illustrate the review results for each of the questions, one by one, supported with statistics from the data extraction.

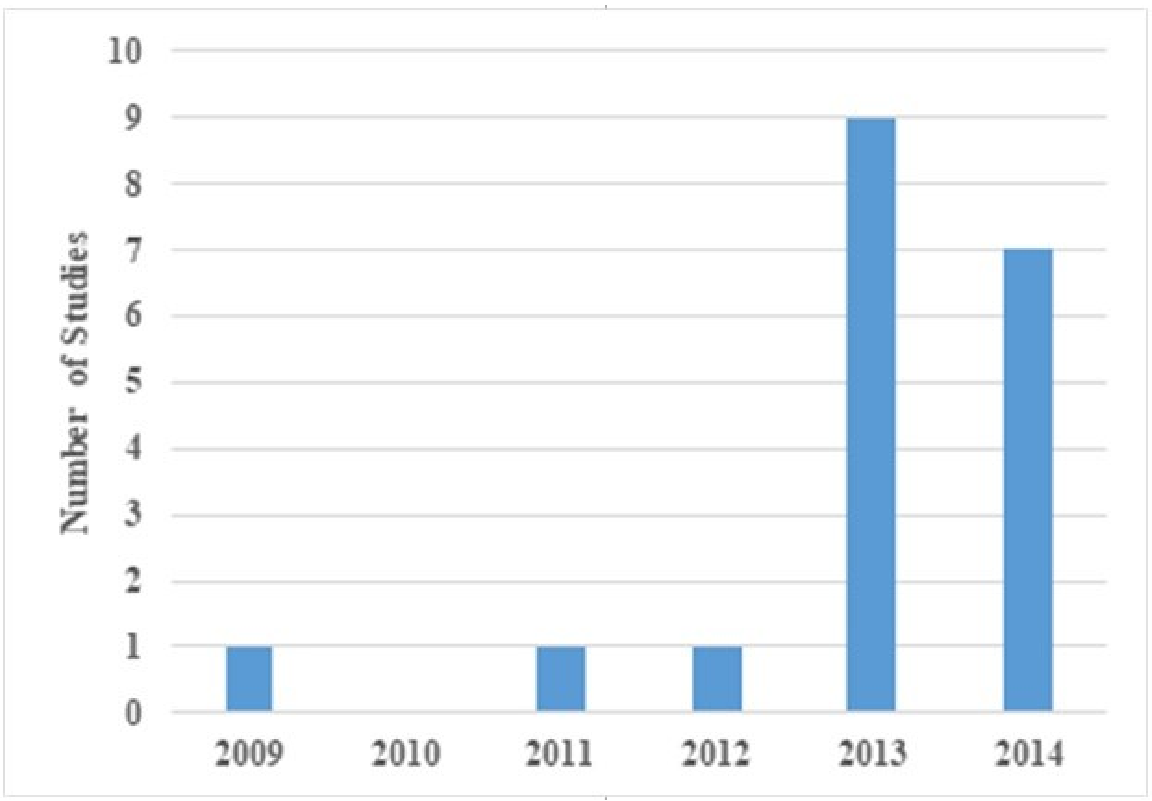

**Figure 2-5 Distribution of Selected Studies per Year**

### 2.2.2.1 Beneficiary Practitioners (Q1)

The first question of this literature review was defined as follows:

***Q1: Which software practitioners does the available SA research target?***

The target of the first question is to figure out the main practitioners who would benefit from the primary SA studies. From the studies selected, we identified that the main practitioners who would be supported by available SA studies are:

- Developer
- Tester
- Project manager (PM)

- Portfolio manager and Senior management

**Table 2-3 Quality Assessment Levels Statistics**

| Quality Levels | # Studies | Percentage |
|---|---|---|
| Very high (85% ≤ score ≤ 100%) | 9 | 22% |
| High (75% ≤ score < 85%) | 6 | 15% |
| Medium (50% ≤ score < 75%) | 4 | 9% |
| Low (0% ≤ score < 50%) | 22 | 54% |

These results are shown in Figure 2-6, where 90% of all studies targeted developers (17 out of 19) with about 47% (9) exclusively supporting developers (for details see Table 2-4). This shows that SA needs more research regarding stakeholders other than developers. Even the available research work that supports other stakeholders, like PMs, is still undeveloped and is similar to the direct statistics and dashboard work. For example, Stroulia et al. (S3) proposed a framework called "Collaboratorium Dashboard" in order to visualize insights extracted from collaborative software development tools. These tools included information related to a team that has worked on a certain project, project artifacts, communication between project stakeholders, and the process followed. Also, the authors integrated their framework with IBM Jazz and WikiDev, which already included integration with SVN, Bugzilla, email, and wikis.

Although the proposed dashboard provided useful information for PMs in a visual form, such as the number of emails sent by each team member and the number of files checked

**Table 2-4 Q1 Extracted Data**

| Practitioner | Supporting Studies |
|---|---|
| Developer | S1, S4, S5, S6, S7, S8, S9, S10, S11, S13, S14, S15, S16, S17, S19 |
| Tester | S2, S13 |
| Project Manager | S2, S3, S4, S8, S10, S11, S12, S13, S18, S19 |
| Portfolio Manager | S10, S19 |

in by each developer, this still formed a straight-forward insight extraction or statistics from software artifacts. More analytics are needed to link more than one artifact and get more supportive and powerful decisions. This can be the link between the source code of a certain feature, the emails related to this feature, or the quality reports, which can be very useful to highlight the need for refactoring a certain part of this code. Such advanced analytics are a major need for any future research in SA.

#### 2.2.2.2 Research Domain (Q2)

The second question of this literature review was defined as follows:

***Q2: Which domains are covered by SA studies?***

The aim of the data extracted for Q2 was to identify the main active SA research domains in order to support practitioners in deciding both innovative and cutting edge topics and research opportunities.

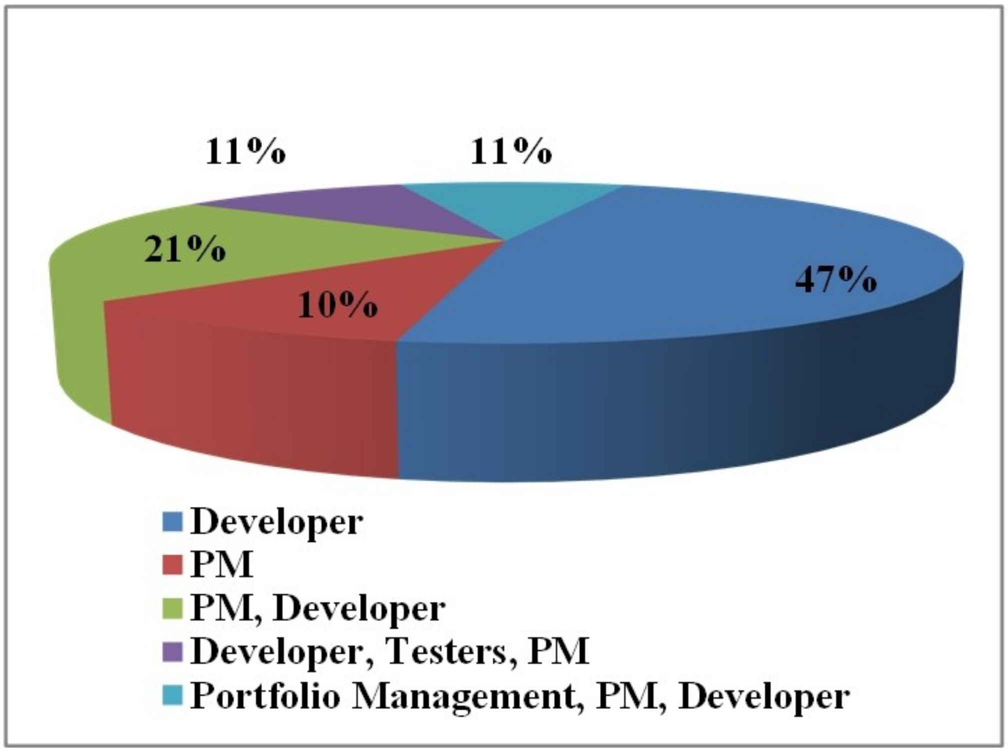

**Figure 2-6 Distribution of Selected Studies per Practitioner**

Our review showed that most available SA studies fell into one of the following domains:

- Maintainability and Reverse Engineering
- Team Collaboration and Dashboard
- Incident Management and Defect Prediction
- SA Platform
- Software Effort Estimation

The distribution of the studies considered per domain can be found in Figure 2-7 (for details see Table 2-5).

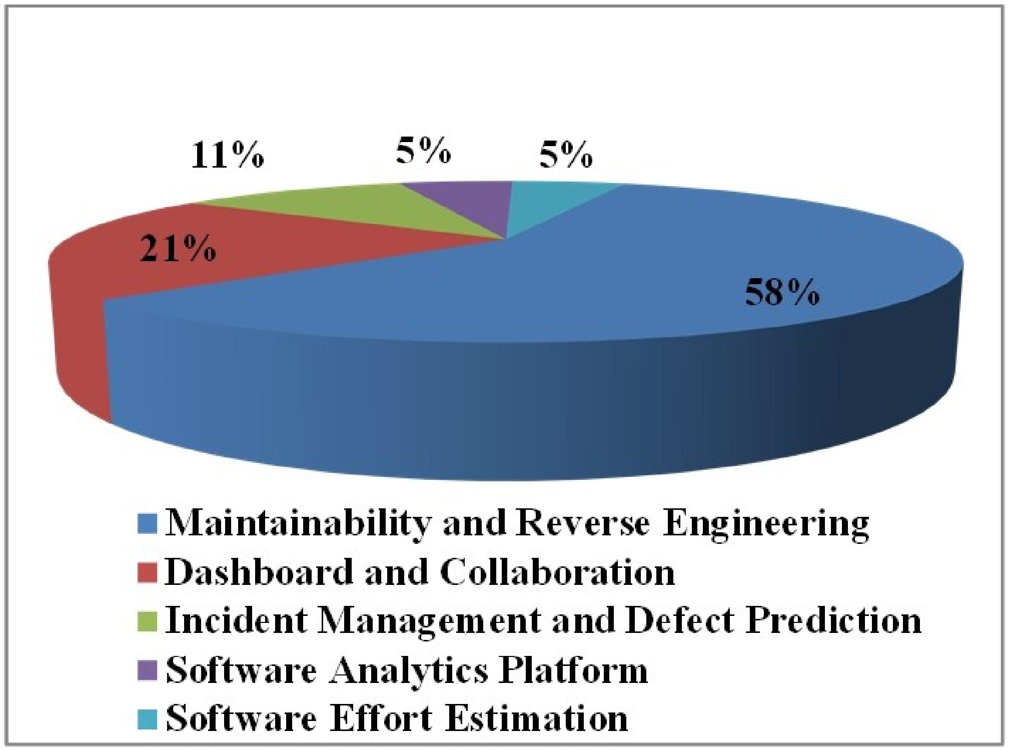

**Figure 2-7 Distribution of Selected Studies per Domain**

**Table 2-5 Q2 Extracted Data**

| **Domain** | **Studies** |
|---|---|
| Maintainability and Reverse Engineering | S1, S2, S4, S5, S7, S12, S13, S14, S15, S16, S17 |
| Team Collaboration and Dashboard | S3, S9, S10, S18 |
| Incident Management and Defect Prediction | S6, S8 |
| Software Analytics Platform | S11 |
| Software Effort Estimation | S19 |

In the following subsections, we illustrate our findings for the most significant studies in each domain.

##### 2.2.2.2.1 Software Analytics for Software Maintenance and Reverse Engineering

Gonzalez-Torres et al. (S2) provided a visualization tool (Maleku) which extracts facts and insights from large legacy software and provides PMs and developers with useful information to support their decisions related to software maintenance. This tool extracts information from software repositories and monitors the repository for any updates in order to redo the analysis.

Although the proposed tool provided visualization features, these features simply represent traditional statistical information, like extracting the metrics related to inheritance and interface implementation.

Another study by Van den Brand et al. (S1), presents SQuAVisiT – a powerful visual SA tool. It has been successfully applied to the maintainability assessment of industry-sized software systems, combining results of metrics analysis (such as quality analysis), and visualization of these analysis results. The tool provides software design metrics such as cyclomatic complexity and inheritance depth. The tool also provides checking of code convention, duplication, and bad practices. Although the visual tool provided is useful, the metrics analysis is traditional and appears in older literature.

Minelli and Lanz (S5) are trying to determine whether the traditional maintainability approaches are valid for mobile applications (apps). They rely on the analytics of three artifacts: source code, third party API invocation, and application revision historical data. Minelli and Lanz implemented a visualization SA tool for mobile apps called "SAMOA" (Software Analytics for Mobile Applications). The tool provides visual presentation for multiple software metrics, apps versions, and the size of relative lines of code between core functionality and third party invocation.

Although the visualization tool presented can support project management, the metrics presented are very similar to traditional metrics from literature. It was expected to use more available artifacts such as user comments and ratings from app stores (like iOS Apple store or Google apps store). Also, Minelli and Lanz rely on only one dataset for their study.

Klammer and Pichler (S7) introduced a reverse engineering tool and applied it to electrical engineering software programs. The tool analyzes only the software source code in order to provide some insights related to source code structure and to locate features within source code. Multiple languages are supported such as C++ and Python. This work is similar to traditional work, and it needs to consider other software artifacts in order to apply SA concepts.

##### 2.2.2.2.2 Software Analytics for Team Collaboration and Dashboard

Baysal et al. (S9) provided the Mozilla development team with a new qualitative dashboard as a complementary tool for the traditional quantitative reports of the Bugzilla issue tracking system. The qualitative dashboard improves development team awareness of the project situation and future directions. New features were provided, such as guiding developers to new information regarding their patches since the last check, highlighting new comments and reassigned patch reviewers. This research is promising since the trend towards qualitative analysis is strong, and it can facilitate and speed up the decision-making process which has traditionally relied on deep quantitative statistical analysis. However, the features provided are very direct and can be easily achieved by reviewing the bug history on the issue tracking system. In order to make this work in a more sophisticated way, new features such as team productivity trend charts can be provided.

##### 2.2.2.2.3 Software Analytics for Incident Management and Defect Prediction

Lou et al. (S6) introduce an SA tool called Service Analysis Studio (SAS). SAS supports engineers in improving incident management by facilitating and automating the extraction of supportive insights. SAS has the ability to use multiple data sources – such as performance counters, operating system logs, and service transaction logs – to provide insights.

What makes this study important is that it applies the SA concept by linking multiple artifacts. Also, it presents a new algorithm to analyze system metrics data and suggests which abnormal metric is suspected of being the root cause of the incident. In addition, it introduces a mining technique to find the suspicious execution patterns—which are the sequence of actions that led to the incident— within the huge number of transaction logs.

##### 2.2.2.2.4 Software Analytics Platform

Czerwonka et al. at Microsoft (S11) provide an SA common platform called CODEMINE. The need for CODEMINE emerged from the observation of the commonality between the input, outputs, and processing needs of multiple analytics team tools. CODEMINE acts as the common analytics framework for multiple client SA applications at Microsoft. The CODEMINE's ability to provide data from different software artifacts (such as source code, project schedule, milestones, and defect reports) opens new research opportunities at Microsoft. In turn, this will enrich insights by extracting information from cross-products which will boost team collaboration.

##### 2.2.2.2.5 Software Analytics for Software Effort Estimation

G. Robles et al. (S19) present a study on the effort estimation of the OpenStack open-source project. Effort estimation of open source projects is challenging, as such projects have both a collaborative and distributed nature, and it is difficult to track the development effort. As a result, the authors offer a model that extracts data related to developer activities from the source code management repository, and then guesses the effort roughly based on these activities (like the time between two commits). Then, the model calibrates the rough estimates based on other estimates collected from the developers in a survey. This study is promising, especially in the way it links artifacts to obtain insights that are useful for tackling such a hard-to-track topic as effort estimation of open source software projects.

#### 2.2.2.3 Analyzed Software Artifacts (Q3)

The third question was defined as follows:

***Q3: Which software artifacts are extracted?***

In order to address Q3, we extracted the analyzed artifacts in each study. This was very important for our study to be able to evaluate the alignment of the studies with the goal of having SA analyze more than one software artifact and provide more advanced insights.

The results of the review show that around 47% of the studies are still using only one artifact (9 studies), and many of these studies only analyze source code, as do traditional software analysis and metrics studies (4 studies). These results support our conclusion that most of the currently available SA studies are still in the early stages and reflect confusion about the difference between direct software analysis and the new SA. The results summary is shown in Figure 2-8; more details can be found in Table 2-6.

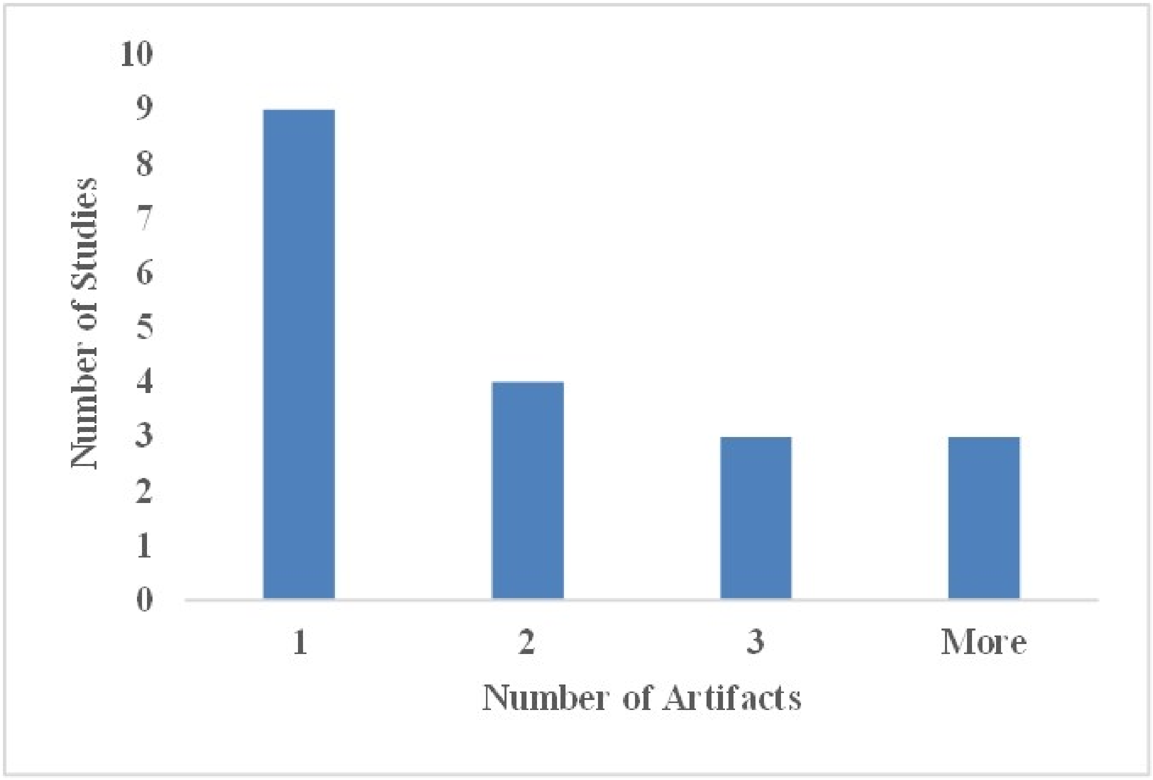

**Figure 2-8 Number of Analyzed Artifacts Versus Number of Studies**

**Table 2-6 Q3 Extracted Data**

| **Study ID** | **Analyzed Artifacts** |
|---|---|
| S1, S4, S7, S16 | Source code |
| S2 | Code repository |
| S3 | Source code repository, issue tracking system, email, wikis |
| S5 | Source code, version control system |
| S6 | Performance counters, operating system logs, service transaction logs |
| S8 | Issue management system, version control system, code reviewing system, source code, organizational data, testing data |
| S9 | Issue tracker |
| S10 | Process data, product data |
| S11 | Source code, project schedule, milestones, defect reports |
| S12 | Source code, bug reports |
| S13 | Source code, version control system, bug reports |
| S14, S15 | Call stack |
| S17 | Mobile apps users reviews |
| S18 | Team wiki, version control system, issue tracking system |
| S19 | Version control system, developers survey |

#### 2.2.2.4 Checking Artifacts Linkage Before Analysis (Q4)

The fourth question was defined as follows:

***Q4: If different artifacts are used, are they linked together?***

In order to address the last question, Q4, we evaluated the analysis of the artifacts used. The main goal was to make sure that the artifacts were linked together in order to get more complex insights that could support software practitioners in making their decisions. It is worthwhile to highlight that this analysis was valid for only 10 studies when more than one artifact was used. This was achieved by reviewing the study scores for the third quality assessment criteria (QA3). The results show that eight studies scored 100%, which reveals that these studies link multiple artifacts to get insights that can support decision making. Therefore, these studies comply with the SA concept and can be considered as good references for practitioners to understand the SA concept. For more details, see quality scores in Table 2-2.

### 2.2.3 Systematic Literature Review Limitations

In this review, we considered both journal and conference papers without evaluating their rankings. This can be attributed to the difficulty that we faced when trying to find well established and relevant papers, which was a result of two factors. The first was that the SA field was less than four years old at the time of this review. The second factor had to do with the misuse of the term SA and the confusion of the researchers about its correct indication. This was shown by the number of papers considered after applying the filtration phases, as previously mentioned.

## 2.3 Systematic Literature Review Addendum

Since SA is an emerging field and the conducted SLR covers research work up to 2014, we re-ran our searching terms on both libraries considered for the period January 2015-August 2018 (the time of defending this thesis). The search included both the metadata and body (content) of literatures for both libraries. The search results contained 176 papers from IEEE Xplore and 141 papers from ACM Digital Library. Since the main objective of conducting the SLR was to come up with open research gaps in the SA research area, we examined the status of the same gaps. This was achieved by analyzing the new search results list, which confirmed the existence of the same research gaps that had been detected by our previous SLR. Those gaps include the analysis of only one software artifact (in most cases is the source code), and the scarcity of studies that target stakeholders other than developers. However, our search detected a few studies (around 8) that have done some promising effort, which comes aligned with our recommendations from the SLR, such as targeting more stakeholders [32] and exploiting more artifacts [33].

## 2.4 Summary

In this chapter, the available SA studies were investigated in order to understand the current status of this new research topic. We conducted a literature review searching for the relevant studies available from 2000-2014. Our review considered 19 primary studies that supported us in addressing the four defined questions. The results can be summarized as follows:

- Q1: The practitioners who benefit from the current SA studies are developers, testers, PM, portfolio managers, and senior management; about 47% of the considered studies support only developers.

- Q2: The studies considered showed that SA research covered the domains of maintainability and reverse engineering, team collaboration and dashboards, incident management and defect prediction, the SA platform, and software effort estimation.

- Q3: Most of the studies considered (around 47%) analyze only one artifact for their study.

- Q4: Most of the studies we considered analyze more than one artifact providing more complex insights, but there is still room for improvement of these studies. The review results showed that most of the available SA research introduces direct software statistics like design metrics and change history, and simply embellish these with some new analytics contributions such as linking team members to the classes they update. Also, most of the research addresses the low-level analytics of source code.

Based on our analysis, this review provides a recommendation for researchers that more research and elaboration needs to be done, such as considering more artifacts in order to add value to traditional work, and using more datasets to achieve higher confidence levels in the results. In addition, there is a lack of research targeting higher-level business decision making, such as project management, portfolio management, marketing strategy, and sales strategy.  This was one of the main triggers for the selection of our research problem, as explained in more detail in Chapter 4.

# Chapter 3

# 3 Lessons Learned Recall Background and Motivation

In this chapter, we will explain the main concepts employed in this research work. This includes the description of the lessons learned (LL) definition and the fundamentals of the information retrieval (IR) models. Moreover, we will present the state-of-the-art and research gaps of the LL recall field will be illustrated.

## 3.1 Lessons Learned in Software Engineering

The LL could be conceived of as an important part of the organization's memory and cumulative experience and knowledge. LL could be guidelines, handling scenarios or tips related to what went wrong (mistakes), or what went right (opportunities), in certain situations or events. In addition, LL could be a success that the organization wants to repeat, or a failure that the organization wants to avoid in the future. The need to preserve the organization's knowledge, which could be lost as a result of several factors, such as expert turnover, calls for the adoption of these LL repositories. The LL concept is evolving, and multiple organizations have their own LL repositories [6] [34].

It is valuable to highlight that LL differ from best practices. In contrast to the best practices that capture only successful scenarios, the LL can capture both success and failure scenarios. Also, best practices are ideas that are recommended on the industrial scale and could be localized to the organization, while LL are organization-oriented and could be globalized to the industrial scale.

It is worth mentioning that although LL records can be related to any software practice, such as project management and development, we focus in this work *only* on LL records related to project management.

LL representation should give information about the context or situation where the lesson learned is applicable, the need to apply the LL actions in order to avoid a problem or to leverage an opportunity, and the recommended actions that can be followed in order to avoid the problem or to leverage the opportunity. Table 3-1 shows an example of a lesson learned represented by three fields, namely context, problem/opportunity, and recommendations. In this example, the development team should be at the customer's premises, which means that issuing an entry visa for the team could be the cause of a planning issue. For this reason, the recommended action is to plan for this ahead, as shown in the LL recommendation section.

**Table 3-1 Lesson Learned Example 1**

| Attribute | Value |
|---|---|
| Context | One of our project constraints is to have the development team onsite (at the customer's site), and our customer is in X country. |
| Problem/Opportunity | Issuing a visitor's visa for our team members takes a lot of time, especially during high seasons. |
| Recommendations | Try to keep your staffing plan updated and make sure it covers 1 or 2 months ahead.<br>Try to start the visa issuing process, for any member, 4-5 weeks ahead of the start date of the planned task at the customer's site.<br>Try to seek your customer's support in getting a long-term visa (example: 6 months) with multiple entries. |

*Obs: some sensitive information regarding customer's identity and country was updated or removed due to the non-disclosure agreement*

Another example of an LL record is shown in Table 3-2. This record concerns a decision about whether to implement a mobile application in-house or to outsource the implementation. It is important to highlight that the LL representation can differ from one organization to another. For example, the LL record can be described as a flat text, as in the case of the dataset employed in this thesis, without using specific attributes or fields. For more LL examples, please refer to Appendix A.

**Table 3-2 Lesson Learned Example 2**

| Attribute | Value |
|---|---|
| Context | Project scope includes an implementation of a small-sized mobile application. This mobile application is not reusable, i.e., it will only be used in this project. |
| Problem/Opportunity | If the mobile application is of a small size, then the organizational process and overhead tasks, such as quality assurance and management reporting, will reduce the profit from implementing the mobile application in-house. |
| Recommendations | Outsource the implementation to an external mobile application specialized company. Contact the purchase team for a trusted partners list. |

## 3.2 Lessons Learned Recall State-of-the-Art

The LL information can only be beneficial if project managers (PMs) refer to it for solving present issues or avoiding expected risks, which is not always the case. Unfortunately, LL are often abandoned due to the lack of knowledge of relevant LL by PMs or due to the need to continuously remind them of the existence of new relevant LL [35]. Although, this can be overcome by PMs manually searching for relevant LL records, this is effort and time costly, especially when searching unstructured information. Also, there could be other reasons for disregarding LL repositories, such as time limitations [35]. This calls for effective and automatic LL recall solutions. By *automatic* we mean that there should be no need for manual searching to facilitate and support frequent references to and exploitation of LL. In this section, we present the related work and state-of-the-art regarding the LL automatic recall.

Most of the available LL research focuses on either the LL process or the implementation of a standalone LL repository system [34]. To the best of our knowledge, there is a paucity of software engineering research addressing LL recall solutions [34].

Harrison [36] has introduced a standalone software LL system. In his implementation, he has tried to improve the efficiency of information retrieval by providing different search options. The system provided the ability to search based on domain, keyword, or repository navigation. However, this does not eliminate the need for users to manually define the search query string. This is different than our proposed solution, as will be described in Chapter 4, which makes use of the existing project artifacts to search for the relevant LL.

Sary and Mackey [37] have introduced RECALL, which is a case-based reasoning (CBR) system. CBR has been employed to improve relevant LL retrieval by users. RECALL work differs from our proposed research in significant ways. First, the employed CBR technique is different from the proposed IR techniques that are presented in this thesis. Second, the RECALL system relies on describing the LL in a case-based question-answer format. This format is difficult to follow for the existing organizations' LL repositories.

To the best of our knowledge, the work of Weber et al. [38] is the only available work that does not require users to fully construct the query string. They introduced an LL retrieval tool called "ALDS," and they embedded this tool in a decision-making tool called "HICAP." They provided an implementation for ALDS within the task decomposition of the project planning phase. However, ALDS differs from our proposed solution in multiple ways. First, ALDS employs the case-based reasoning (CBR) technique, while our solution employs the IR approach, which is different, as will be explained in Chapter 4. IR and CBR are different in some aspects; e.g., in CBR, cases are stored in a "case representation" format, where additional inferred knowledge can be kept to make them more fitting for reasoning and learning in new situations [39], while IR relies on searching within the original format of the document repository including all features and terms. The second difference is related to LL similarity evaluation; ALDS relies on the indexing of LL in a question-answer format, where users have to go through answering the questions while describing their task condition. In contrast, this limitation is not required for our solution since it relies on automatically querying the LL classifiers or the search engine using data extracted from the project artifacts. The queries, the issue or risk records, are extracted from the existing project artifacts, which are issue/risk register documents. We describe the proposed methodology in more detail in Chapter 4.

## 3.3 Information Retrieval Models Applied to Recall Lessons Learned

IR refers to the process of finding a relevant document or information of interest within a collection of documents or artifacts. In this thesis, we use the IR term to refer to text IR in mining software repositories. Usually the data within the searched collection, in the case of IR, is in an unstructured format (i.e., natural language text) [40]. The input to the IR classifier is a query, or question, and the result is a list of the documents relevant to this query [41]. The IR idea is similar to that of web search engines, such as Google, where the user provides a query, describing the need or the question, and the search engine tries to answer the user's question by replying with a list of the most relevant web content.

Regarding the employment of IR techniques in solving software engineering problems, it is worth mentioning that IR models have been used to solve several problems in the software engineering domain, such as bug localization [42][43][44] and concept location [45], but have not been employed to improve the LL recall as per our knowledge [46]. Thus, to the best of our knowledge, we are the first to employ IR techniques to solve the LL recall issue within the software engineering context [46]. We will explain our solution in detail in Chapter 4.

There are multiple IR models that can be used to construct classifiers, and they vary based on their theories, such as simple keyword matching and statistics. There are two main factors which affect the operation and the accuracy of the IR classifier. The first factor is the preprocessing steps, which are employed to process the text inputs before forwarding them to the IR classifier. In our case, the text inputs include both the LL records, which are used to construct the IR classifier, and the issue/risk records, which are used to query the constructed classifier. Different preprocessing steps from the natural language processing (NLP) literature can be used. Later in this section, we provide some details regarding the preprocessing steps used. The IR model parameters are the second factor. Each of the IR models or techniques has its own specific parameters which drive the classifier construction and operation. Examples of these parameters can be the similarity, the method to calculate the document relevance to the query, and the term weight. The fact that the impact of parameter configurations on the IR classifiers' performance has already been studied in software engineering domains other than LL recall, such as bug localization [42] and equivalent requirements [47], plays a role in the motivation to consider the parameter configurations impact in our own study. In addition, an optimization of the IR model configurations based on the dataset at hand is important and is an open research area (for more details, please see Section 7.1).

In the following subsections, we will provide some details regarding the preprocessing steps that we applied to the text inputs. Then, we will give an introduction to three of the most popular IR models from the literature that were used in this study. These models are: Vector Space Model (VSM), Latent Semantic Indexing (LSI), and the Latent Dirichlet Allocation (LDA).

### 3.3.1 Text Preprocessing Steps

Since both the documents' corpus, LL repository in our case, and the query comprise unstructured information, they are preprocessed before being forwarded to construct or query the LL classifiers. The preprocessing plays a key role in reducing any information noise, which could be a source of confusion to the LL classifiers. It is common practice in IR research to apply one or more preprocessing steps from the NLP literature [41]. The following is a brief description of the two preprocessing steps we applied in this study:

- Stopping step: removing the common stop words from the English language, such as "the" and 'a'. Such words are very common and have high appearance frequency within the document, which can impact the relevance score while not representing a real relevance of the document to the query.

- Stemming step: reducing the words to their morphological roots or stems. For example, "stem" is the root for both "stemming" and "stems".

### 3.3.2 Vector Space Model

The VSM is an algebraic IR model [40][41]. It relies on representing the documents' corpus in a matrix format of terms versus documents (t x d matrix). In this matrix format, each term in the corpus vocabulary, where the vocabulary contains all the different terms, has a term weight value corresponding to each document in the corpus. The row dimension value of the matrix represents the number of the different terms, where each row represents a term. On the other hand, the column dimension value represents the number of the various documents in the corpus. In each term row, the term has a non-zero weight value if the term exists in the corresponding document, and a zero value otherwise. The term can represent a single word and its weight can be calculated as a simple existing Boolean value, where existing Boolean value ‘1’ is “exist” and ‘0’ is “not exist,” in each document. In order to decide if two documents, or a document from the corpus and a query, are relevant, the VSM model compares these two documents’ columns or vectors from the terms versus the documents’ matrix. This comparison is achieved using a configured similarity method which can be, for example, the inner product of the two documents’ vectors. To consider two documents relevant, they should have one or more common terms. The VSM model returns a proportional continuous similarity value according to the number of common terms between the two compared documents.

The VSM model has two main configurable parameters:

- *Term weight*: the term weight in a document. The basic weight method is the *Boolean* method whose value is ‘1’ if the term appears in the document, and ‘0’ otherwise. Other popular weighing methods are term frequency (*tf*), which is the number of times the term appears in a document, and term frequency-inverse document frequency (*tf-idf*), which is an extended version of the original *tf* with the consideration of the term popularity in corpus documents [40]. For *tf-idf*, the term weight for a certain document is high if it appears with high frequency in this document and, at the same time, the term is rare and has a low frequency within the overall document corpus.

- *Similarity*: the method used to calculate the similarity degree between two document vectors, or, as in our case, between a document and a query. Popular similarity methods include *cosine* distance and *overlap* methods [40].

### 3.3.3 Latent Semantic Indexing

The LSI model is an extension of the VSM model. Unlike VSM, LSI takes the context or topic into consideration instead of only matching the terms, which can have different meanings, polysemy, within different topics. For LSI, documents sharing the same topics, even if they do not share the same terms, can be considered similar documents. This is very important in the case of synonymy and polysemy [41][48]. To achieve this goal, LSI employs a technique called singular value decomposition (SVD). SVD decomposes the term-document matrix (t x d), used by VSM, into three new matrices: the term-topic or T matrix (t x k), the diagonal eigenvalues matrix S (k x k), and the topic-document matrix D (k x d). The k value represents the number of topics, which is a value provided by the model user. The SVD technique works on reducing the rank of both T and D matrices to the provided k value [49]. During this decomposition, the SVD technique works on grouping the co-occurring terms, which appear together, into one topic.

The LSI has three parameters as follows:

- *Term weight*: the same as in the VSM model.
- *Similarity*: the same as in the VSM model.
- *K* or *number of topics*: the number of topics remaining after the SVD reduction.

### 3.3.4 Latent Dirichlet Allocation

The LDA is a generative probabilistic model [41][50]. LDA considers the context of terms by eliciting the topics within the documents' corpus. For the LDA model, each document can be composed of one or more topics with a different membership degree for each topic. Also, the topics can be constructed from one or more terms. Each term can belong to one or more topics with a certain membership value [41].

LDA model has several parameters which can be listed as follows [41]:

- $\alpha$: the document-topic smoothing parameter for the probability distribution.
- $\beta$: the term-topic smoothing parameter for the probability distribution.

- *Similarity*: the same as in the VSM model.
- *K* or *number of topics*: the number of topics to be created by the LDA model.
- *Number of iterations*: the number of iterations considered for the inference process.

## 3.4 Summary

In this chapter, we provided a brief introductory background to the main topics that are addressed in this thesis. We clarified the meaning of LL, and explained how they can be considered as the organization's memory. The value of LL records was underscored by clarifying that they can provide information regarding either historical problem solving or previous opportunity leveraging.

Also, we illustrated that despite the LL repository being a valuable source of knowledge for project managers, it can be abandoned for various reasons, including the difficulty and time required for manual searching of the LL unstructured data. This limitation called for the need for automatic LL recall solutions. Therefore, we explained how such solutions would eliminate the need for manual involvement of project managers to search for relevant LL records, and thus would improve the LL exploitation. We presented the state of current research of the LL automatic recall.

Furthermore, the main concept of the IR techniques was illustrated. Moreover, we described the fundamentals of three of the most popular IR models, which are used in this dissertation, namely Vector Space Model (VSM), Latent Semantic Indexing (LSI) and Latent Dirichlet Allocation (LDA). For each of these models, we clarified the main theory and listed the main configuration parameters.

# Chapter 4

# 4 The Design of the Lessons Learned Recall Solution

Our literature review, from Chapter 2, revealed the existence of research gaps regarding the practitioners who benefit from existing software analytics research. As has been shown, most of the available research supports software developers, while rare studies serve other stakeholders such as project managers. This was the main motive behind focusing our research on addressing this gap by supporting other stakeholders. We took this into account when selecting and defining the problem statement and research goals.

In this chapter, we start by stating the research problem. Then, the translation of this problem into the research questions is explained. In order to solve the problem at hand, we provide a novel solution which is explained in detail. Moreover, we describe how our solution is evaluated and the research questions are answered by conducting an empirical case study. The detailed plan and methodology of this empirical study are also illustrated in this chapter.[2]

## 4.1 Problem Statement

As we described in Chapter 3 (Section 3.1), the lessons learned (LL) records constitute the software organization's memory of successes and failures. The LL are recorded in the organization's repository for future reference in order to optimize planning, gain experience, and elevate market competitiveness.

However, the LL repository is often disregarded despite the valuable information it provides. This can lead to the repetition of previous mistakes, or even missing potential opportunities. This, in turn, can negatively affect the organization's profitability and competitiveness. Disregarding the LL repositories could be due to the lack of knowledge

[2] Part of this chapter is under review in the Information and Software Technology Journal.

**Tamer Mohamed Abdellatif**, Luiz Fernando Capretz, and Danny Ho, "Automatic Recall of Software Lessons Learned for Software Project Managers," *Inf. Softw. Technol. (IST)*, March 2018. (Under review) of relevant LL by project managers (PMs), or to the need to continuously remind them of the existence of new relevant LL [35]. Although, this can be partially overcome by manually searching for relevant LL records by PMs, the effort is labor and time costly, especially when searching in unstructured information. Also, there could be other reasons for abandoning LL repositories, such as time limitations [35].

## 4.2 Research Questions and Goals

Based on the defined problem statement, it is clear that our research targets PMs, which clearly aligns with the research trigger of supporting different stakeholders than developers in order to close one of the research gaps from our systematic literature review (SLR).

The primary objective of this research is to leverage the benefits to PMs that can be gained from the organization's LL knowledge. We believe that this can be achieved by both facilitating the retrieval of relevant information and boosting knowledge about relevant and useful LL. We aim to achieve this by employing information retrieval (IR) techniques to provide adequate automatic LL retrieval classifiers. Also, our solution relies on constructing the search query automatically from existing project artifacts (issues and risks) as described in detail in the methodology section (Section 4.3). This closes another gap from the SLR since we exploit software artifacts other than source code. To the best of our knowledge, we are the first to employ IR techniques in mining the software LL repository; this is consistent with the literature survey conducted by Chen, Thomas and Hassan [46].

In order to achieve our objective, we have defined three main research questions which we answer in this research. The research questions are as follows:

*RQ1: Can we automatically, rather than manually, recall and push the relevant LL to PMs using IR-based LL classifiers?*

This research question is two-fold. In the first part, we are examining whether having an automatic LL recalling solution is efficient. This is important for solving the main problem of having the intensive process of manually searching the LL repository for relevant records. Since we plan to employ IR techniques to provide our solution, the second part of this research question examines the fitness of IR techniques for the LL recall context.

Reaching an answer to this research question is achieved by examining the performance of the solution through an empirical case study as described in Chapter 5.

*RQ2: Can project artifacts be used to construct on-the-fly queries to recall LL records relevant to the project at hand?*

As the main characteristic of our solution is to be automatic, i.e., no manual search by PMs, this research question examines the effectiveness of constructing the search query on-the-fly from existing project artifacts, as described in detail in Section 4.3.2, instead of relying on the manual inputting of the search query by PMs.

The effectiveness of this on-the-fly search query construction can be measured by the ability of the LL classifier to return relevant LL records for the project at hand, i.e., the project to which the artifacts belong. This will be answered by the empirical case study conducted.

*RQ3: Do the configurations of the LL classifiers have an impact on the performance results?*

In this research question, we are seeking a statistical validation of the impact of the classifier's parameter on the performance. We will examine the impact of considering different classifier's parameters on the performance. This is crucial for determining whether the result conclusions and insights are statistically significant, and whether they can be generalized within the dataset at hand and experimental environment context.

This research question is answered by applying an appropriate statistical test to the performance results recorded for the empirical study, as presented in Chapter 5.

## 4.3 Proposed Solution and Case Study Methodology

As described, the intensiveness of manually searching the unstructured LL repository for relevant records is a major cause of the LL abandoning problem. The main contribution of our solution is to address this problem by replacing the manual search with an automatic search for relevant LL. Thus, in our solution, there is no need for the manual involvement of PMs in constructing the search query. Instead, we rely on existing project artifacts to build the query string on-the-fly. Moreover, another contribution of this work is employing IR models for the first time, as per our knowledge, to construct an IR-based LL classifier. We chose IR models because they have shown superior results for similar problems, in the software engineering literature, such as bug localization and equivalent requirements.

The LL classifier, in our solution, is able to retrieve a list of the LL records relevant to the current project a PM is working on. The classifier provides these relevant results based on a query that is automatically generated from existing project artifacts extracted from the project repository.

In the rest of this section, we describe in detail each part of this solution, including the methodology employed to construct the automatic LL classifier solution. Also, the methodology employed to conduct the empirical study, in Chapter 5, is defined. This covers the processes of constructing the LL classifier, the search query, and the evaluation process.

### 4.3.1 Lessons Learned Classifiers Construction

As we mentioned, we rely on IR models in constructing the LL classifiers. Accordingly, three popular IR models from the literature, are employed, namely, Latent Semantic Indexing (LSI), Latent Dirichlet Allocation (LDA), and Vector Space Model (VSM). In this study, we will construct multiple classifiers and compare their performance in order to identify the most effective classifier for the problem at hand. To construct these classifiers, we have to define three types of configurations: data representation, preprocessing steps, and model-based parameter configurations.

#### 4.3.1.1 Data Representation Configuration

The data representation configuration defines which parts or fields from the search query and LL record will be employed while calling or constructing the LL classifier.

As will be described in detail in Chapter 5, both the project artifacts (search query), and the LL records are represented using their full description field values. We only rely on the description field value because other fields, such as “title,” were not defined for the dataset provided.

#### 4.3.1.2 Preprocessing Steps Configuration

The preprocessing configuration defines how data (project artifacts and LL records) is preprocessed before being forwarded to the IR algorithm to build the LL classifier. Since the selection of the appropriate preprocessing steps is an open research area [41], for our case study, we have chosen to employ two of the most common techniques from the natural language processing (NLP) literature, namely: *stemming* and *stopping*. In *stemming*, the words are reduced to their word stem. In *stopping*, the stop words are removed from the original text. In order to apply these two preprocessing steps, we use the tool provided by Thomas [51]. We consider the four combinations of applying these two preprocessing steps: not applying any of the two steps (none), applying *stemming* individually, applying *stopping* individually and applying both *stemming* and *stopping*.

#### 4.3.1.3 Model-Based Parameter Configuration

For the LSI model, there are three parameters which should be configured: *number of topics*, *term weight,* and *similarity*. Since there is no optimal selection method for the *number of topics*, and since it is still an open research topic, we consider four values from the literature [42] for *number of topics*; “32,” “64,” “128” and “256.” Those chosen values cover the different ranges of the *number of topics* values [52]. Regarding *term weight*, we consider three methods from the literature [40], namely: the *Boolean*, *tf-idf*, and *sublinear tf-idf* methods. For *similarity*, the *cosine* similarity method is employed, as it is the most suitable method from the literature for the LSI model [40][42].

For the LDA model, we consider the same *number of topics* values as in LSI. Other parameters, including *sampling iterations number*, *topic-word smoothing*, *document-topic smoothing*, and *similarity*, are automatically optimized by the MALLET tool [53], which we use for the case study experiments (More details in Section 5.2). Also, for the query execution, we used the lucene-lda tool, which is implemented by Thomas [54]. The lucene-lda tool employs the *conditional probability* method for the *similarity*, as it is the most appropriate similarity method for the IR applications [42][55].

Regarding the VSM model, there are two parameters: *term weight* and *similarity*. For *term weight*, we employ the same methods as in the LSI model. For *similarity*, we consider both the *cosine* and the *overlap* methods from the literature [40].

#### 4.3.1.4 Overall Configurations Considered

We consider a fully factorial design [42] for the case study experiments, which means that we consider all combinations of the selected parameter values, i.e., data representation, preprocessing steps and model-based parameters. So for each parameter, every value considered is examined against all values of all other parameters.

Accordingly, our experiment has yielded **88** LL classifiers; **48** LSI classifiers ((1 project artifacts representation) * (1 LL records representation) * (4 preprocessing combinations) * (4 number of topics values) * (3 term weighting methods) * (1 similarity method)), **16** LDA classifiers ((1 project artifacts representation) * (1 LL records representation) * (4 preprocessing combinations) * (4 number of topics values)), and **24** VSM classifiers ((1 project artifacts representation) * (1 LL records representation) * (4 preprocessing combinations) * (3 term weighting methods) * (2 similarity methods)). We have tested and evaluated all of these classifiers.

A summary of the LL classifiers construction process is shown in Figure 4-1. Also, the considered parameter values are summarized in Table 4-1.

**Table 4-1 Parameter Configurations**

| Parameter | Value |
|---|---|
| *Common parameters* | |
| Preprocessing steps | None, Stemming, Stopping, Stemming and stopping |
| *VSM model parameters* | |
| Term weight | tf-idf, sublinear tf-idf, Boolean |
| Similarity | Cosine, overlap |
| *LSI model parameters* | |
| Term weight | tf-idf, sublinear tf-idf, Boolean |
| Number of topics | 32, 64, 128, 256 |
| Similarity | Cosine |
| *LDA model parameters* | |
| Number of topics | 32, 64, 128, 256 |
| Number of iterations | Until model convergence |
| Similarity | Conditional probability |

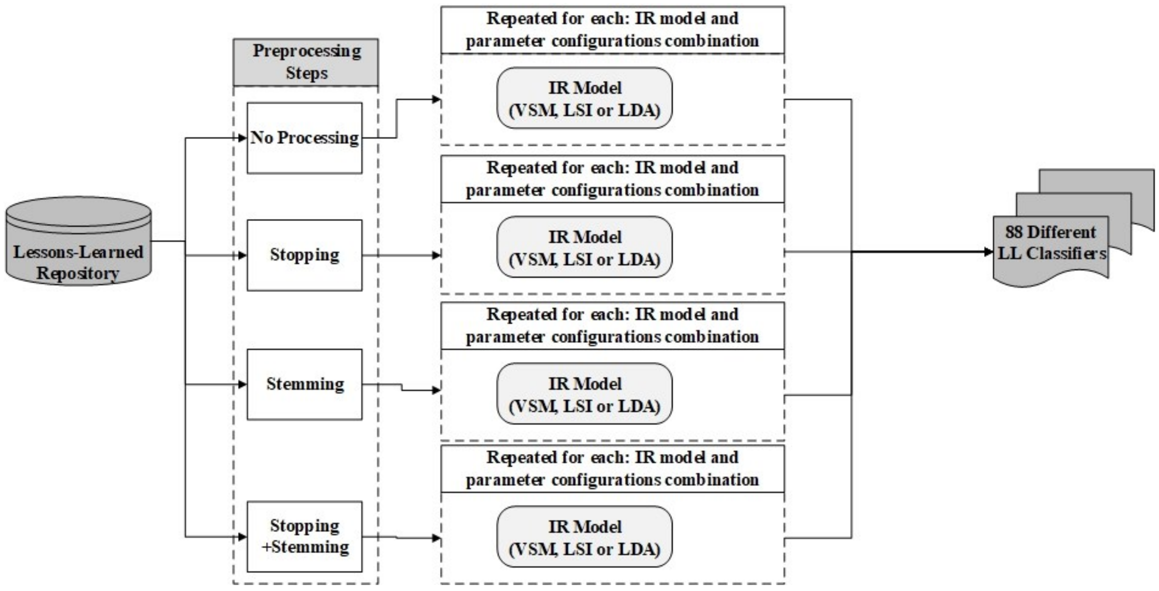
Lessons-Learned Repository
Preprocessing Steps
No Processing
Stopping
Stemming
Stopping +Stemming
Repeated for each: IR model and parameter configurations combination
IR Model (VSM, LSI or LDA)
88 Different LL Classifiers

**Figure 4-1 Construction of the Lessons Learned Classifiers**

### 4.3.2 Dynamic Query Construction

As we described, our solution relies on automatic construction of the search query. To achieve this, in the case study, we employ two types of the available project artifact records, namely issue records and project risk register records, to dynamically construct the query string on-the-fly and search the constructed classifiers. It is important to clarify that by issue records we mean project management issues, such as cost management and team management issues, and not development issues or bugs. Examples for both issues and risks are provided in Appendix A.

By using the existing artifacts, we bypass the need for users to manually construct the query string, and we provide an automatic search process.

## 4.4 Evaluation Process

For the evaluation process, we follow the Cranfield evaluation methodology [56]. This methodology is suitable for the empirical evaluation of IR models. For this evaluation method, we first need to acquire a real dataset, including both the LL and the project artifacts (issues and risks). Therefore, we contacted multiple industrial partners in order to collect the needed dataset for the evaluation. We successfully reached an agreement with one of our industrial partners to provide us with the needed dataset. Also, for the evaluation, we need to build a gold set. The gold set should contain a mapping set of each query examined and the relevant results expected for this query. The detailed data collection and the construction of this gold set are described in Section 5.1 of the next chapter. This set can then be reused to evaluate multiple LL classifiers. The evaluation process is conducted based on defined performance metrics.

Having both the dataset and the gold set, we then pursue the evaluation process by applying the data preprocessing steps, following the preprocessing combinations, as described in Section 4.3.1.2, to the LL repository in order to get different preprocessed versions of the repository. We build the LL classifiers based on each of the LL repository versions, and then we repeat this for each of the IR model configuration combinations that we have considered in Section 4.3.1.4 (see Figure 4-1).

After building the classifiers, we execute each of the queries considered, i.e., issues and risks, using each of these classifiers, and then we record the results list. The performance metrics are calculated, as described in the next section, for each classifier by comparing the results list to the gold set. A summary of the evaluation process is shown in Figure 4-2.

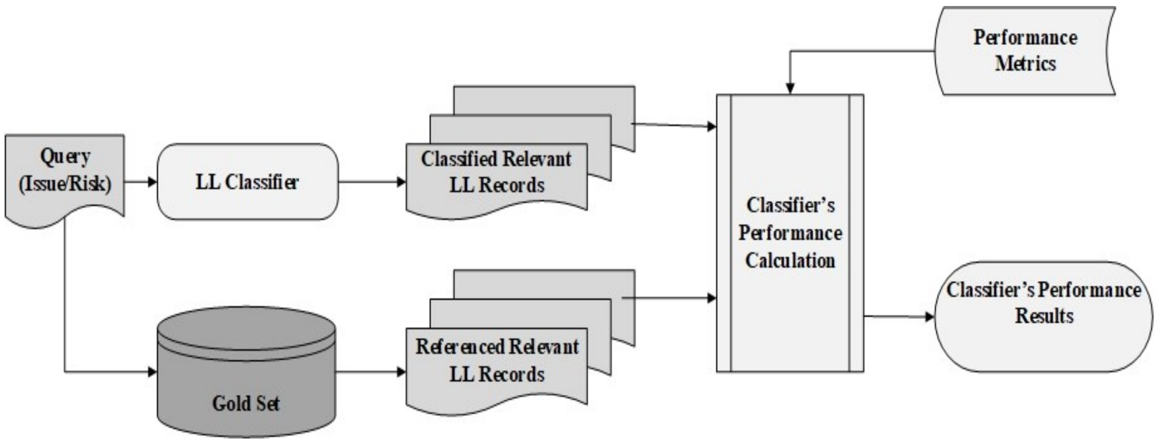
Performance Metrics
Query (Issue/Risk)
LL Classifier
Classified Relevant LL Records
Classifier's Performance Calculation
Classifier's Performance Results
Gold Set
Referenced Relevant LL Records

**Figure 4-2 Lessons Learned Classifier Evaluation Process and Performance Results Calculation**

*Obs: this process is repeated for each classifier, and is calculated for each of the queries' results. Then, the average performance metric is calculated for each classifier*

## 4.4.1 Performance Metrics

To benchmark the performance results for each of the LL classifiers considered, we employ two of the most popular performance metrics from the literature [40][41][42], namely, top-K and MAP (Mean Average Precision). The top-K accuracy metric calculates the percentage of queries, i.e., project issues/risks, whose top k result records have at least one LL record relevant to this query, based on the gold set. The top-K value is significant to our case study because it measures the ability of the LL classifier to provide users with at least one relevant result in an advanced position in the results list, which is important to encourage users to use the new searching tool; this can lead to improvements in the organization's LL recall – our main goal. In the study, we follow the literature by setting k to 20 in order to measure the accuracy of the classifiers when considering the top 20 records from the relevant records retrieved. In the literature [42][43], the value of 20 has been justified as a convenient number of result records through which the user can scroll down before disregarding the search results. Top-K calculations can be formulated as follows [42]:

$$top - K(C_i) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} I\left(Relevant\ docs\ (q_j), TopK\ Records\ (C_i, q_j, k)\right),$$

where $C_i$ is the classifier *i,* $|Q|$ is the total number of the queries examined, $Relevant\ docs\ (q_j)$ is a function that returns all of the relevant documents to the *j*th query based on the gold set, $TopK\ Records\ (C_i, q_j, k)$ is a function that returns the top $k$ result records from the retrieved list for the $q_j$ by the *i*th classifier $C_i$, and finally *I* is the intersection function which returns '1' if there is at least one common document between the two document lists returned by the two functions, $Relevant\ docs$ and $TopK\ Records$, and returns '0' otherwise.

In the case study, the query can have more than one relevant LL record, so it is important to measure the ability of the constructed LL classifiers to recall all possible relevant records, as well as evaluate the classifier's retrieval precision. In order to fulfill this measurement, we employ the MAP metric, which is one of the most popular and most appropriate metrics, from the literature [40][56] for this kind of measurement, especially when comparing multiple IR models and with the existence of multiple query sets. The MAP metric can be calculated as the average of the aggregated average precision of each individual query. The MAP equations are formalized by Zhai and Massung [56] as follows:

$$MAP(\mathcal{L}) = \frac{1}{m} \textstyle\sum_{i=1}^{m} avp(\mathcal{L}_i),$$

$$avp(\mathcal{L}_i) = \frac{1}{|Rel|} \textstyle\sum_{j=1}^{n} p(j),$$

where $\mathcal{L}_i$ is the ranked results list returned by the classifier to answer the *i*th query from the different *m* queries considered; $avp(\mathcal{L}_i)$ is the average precision for the ranked list $\mathcal{L}_i$.

The *avp* is calculated for each query, based on the above equation where $p(j)$ is the precision at the ranked record $j$ within the results list $\mathcal{L}_i$, $Rel$ is the set of all documents relevant to this query based on the mapping set, the gold set, and n is the count of the records of the results list $\mathcal{L}_i$. $p(j)$ is ‘0’ if the $j$th document is not relevant to the query. On the other hand, if the document is relevant to the query, then $p(j)$ will be calculated by dividing the number of relevant documents, identified relevant so far, by the document rank, i.e., j value. For example, if the seventh document within the results list is the fourth relevant document retrieved, then $p(7) = \frac{4}{7}$.

### 4.4.2 IR Configuration Impact

As we planned to study the impact of the different parameter value configurations, i.e., preprocessing steps and parameter values of models, on the classifiers’ performance, we have applied the Tukey’s Honestly Significant Difference (HSD) statistical test [57][58][59]. The HSD test is a statistical test which has the ability to perform a comparison between different groups in one step. The advantage of the Tukey’s HSD test is that it can significantly differentiate between more than two groups based on the statistically significant difference between the groups’ mean. For our study, we use the HSD test to statistically compare the impact of the different parameter configurations on the classifier performance. We studied that parameter by parameter.

So, for each parameter (e.g., *term weight*), we compare the different performance results of each parameter value (e.g., *tf-idf* versus *sublinear tf-idf* versus *Boolean*). While studying a certain parameter, the other parameters may vary. The HSD test examines the difference in the mean value between the results of the parameter value pairs. For each of these pairs, if the difference between their mean values exceeds the expected standard deviation, then HSD can report these two parameter values as statistically different groups. Therefore, any two parameter values can be either statistically different, i.e., reported as different groups, or not statistically different, i.e., the same group, based on the mean difference. Also, any parameter value can belong to one or more groups.

## 4.5 Summary

In this chapter, we started by describing the research problem. We illustrated the existing problem – that PMs have been abandoning the LL repository. We noted that this has been mainly attributed to the intensive task of manually searching this unstructured repository. In order to address this problem, we defined the research questions and illustrated the rationale and motivation behind considering each of them. Moreover, we discussed our automatic LL recall solution and identified the research contributions. The first contribution is employing IR models, for the first time, to construct an LL recall classifier. The second contribution is providing an automatic LL recall solution in order to avoid the intensive manual searching that is currently required to locate relevant lessons learned. We have clarified that our solution is automatic, i.e., no need for a manual search, since the existing project artifacts are employed to construct the search query on-the-fly.

We clarified that answering the research questions and evaluating the effectiveness of the LL recall solution are achieved by conducting a real empirical case study. The empirical case study methodology was illustrated. This methodology includes the process of constructing the LL classifiers using three of the popular IR models. Also, the different types of parameter configurations we considered were clarified in detail. The project artifacts that were employed have also been described.

At the end of this chapter, the details of the evaluation process were clarified, which included the construction of different classifiers, and the evaluation of the LL list retrieved by the classifiers against the expected list based on the gold set. Furthermore, the two performance metrics employed, namely top-K and MAP, were demonstrated. This included the selection rationale and the calculation procedure for each of these two metrics. In addition, we described how the Tukey's Honestly Significant Difference (HSD) statistical test was employed to study the statistical impact of the different parameter configurations on the LL recall classifiers.

After providing the detailed plan for the case study in this chapter, the details of how the case study was conducted will be illustrated in Chapter 5. This includes the detailed results, observations and main findings from the case study.

# Chapter 5

# 5 Can Lessons Learned Be Recalled Automatically: An Empirical Study

In this chapter, we describe the execution steps of the case study plan illustrated in Chapter 4. This chapter starts by describing the data collection process, including the data disclosure agreement, the dataset description and the process that will be followed to construct the reference gold set. Moreover, brief technical details are given regarding the tools employed to construct information retrieval (IR) based classifiers. In addition, the study results, findings and threats to validity are discussed in this chapter. Both the performance results and the analysis of the parameter configuration's impact on the classifiers' performance are discussed. The performance results of all the lessons learned (LL) classifiers that are considered are presented and grouped based on the performance metric employed, either top-k or MAP.[3] For more details regarding the performance metrics considered and the configuration impact analysis plan, refer to Chapter 4, Section 4.4.

## 5.1 Data Collection

One of the most challenging steps for the success of this case study was to collect the dataset needed to evaluate and answer the research questions. Keeping in mind the need for confidentiality and the competitiveness that exists within the software industry, it was not an easy task to get access to the needed dataset, especially given that we targeted real industrial records.

After communicating with our industrial network, we successfully received the data needed from an industrial partner which is a large and reputable multinational software company with a workforce of 800+ employees. Our industrial partner is both ISO 9001

[3] Part of this chapter is under review in the Information and Software Technology Journal.

**Tamer Mohamed Abdellatif**, Luiz Fernando Capretz, and Danny Ho, "Automatic Recall of Software Lessons Learned for Software Project Managers," *Inf. Softw. Technol. (IST)*, March 2018. (Under review) and CMMi Level 3 certified, with more than 20 years in the global IT services domain. The company has global branches all over the world, including North America, Canada, and Arab Gulf countries, such as the United Arab Emirates, Saudi Arabia, and Kuwait. The company provides software solutions within seven different industries, including telecommunications, banking, education and government sectors, in addition to strategic education programs and partnerships with multiple Arab Gulf governments, including Dubai and Qatar governments.

### 5.1.1 Dataset Description

The data provided is under a non-disclosure agreement. Accordingly, the dataset records have been made totally anonymous by the partner by removing all sensitive data, such as customer names and project names. The collected dataset consists of two parts. The first part is the LL repository, while the second part is the project issues/risk register documents.

The LL repository sample provided contains 212 LL records from 30 different software projects. Each LL record is represented by both the project's identification number field, identifying the project which has reported the LL, and the description field. The description field contains a description of the LL and its context in a flat text format.

Regarding the project issue/risk records, we have received 55 issue/risk records from five different projects that are different than the 30 projects used for the LL records. Those records acted as the query string for our case study. The projects are from different domain verticals, including governmental, management consultancy, educational, and telecommunications projects. The scopes of these projects include migration or new implementation of portals, business processes automation, and learning management systems (LMS). These projects follow either waterfall or iterative development methodologies. Also, the customers represented in these projects are from different countries. All the dataset records are written in English.

### 5.1.2 Gold Set Construction

As described in the case study evaluation process in Chapter 4 (Section 4.4), constructing a reference gold set has been a must for comparing and benchmarking the performance of the different LL classifiers considered in the study. The gold set should contain a mapping set of each query examined and the relevant results expected for this query.

In order to construct this gold set, each of the provided issue/risk records was mapped to the relevant LL records from the LL repository. As this map could be subjective based on the users – practitioners and project managers (PMs) in our case – of the retrieval model, we adopted a procedure similar to the one recommended by Kitchenham et al. [12] in performing data extraction while conducting a systematic literature review (SLR) in the case of having a single main researcher. So, the initial mapping was conducted by the main author, the single main researcher in our case who is a subject matter expert (SME). Then, a review meeting was scheduled with another SME from the partner software company. In the review meeting, the company SME reviewed the mapping of the issues/risks to the relevant LL records. In the case of disagreement, the two SMEs held discussions until consensus was reached. After finalizing and agreeing on the whole mapping set, it was baselined. This final mapping set was used for the evaluation and benchmarking of the different LL classifiers within this case study. We summarize the gold set construction process in Figure 5-1.

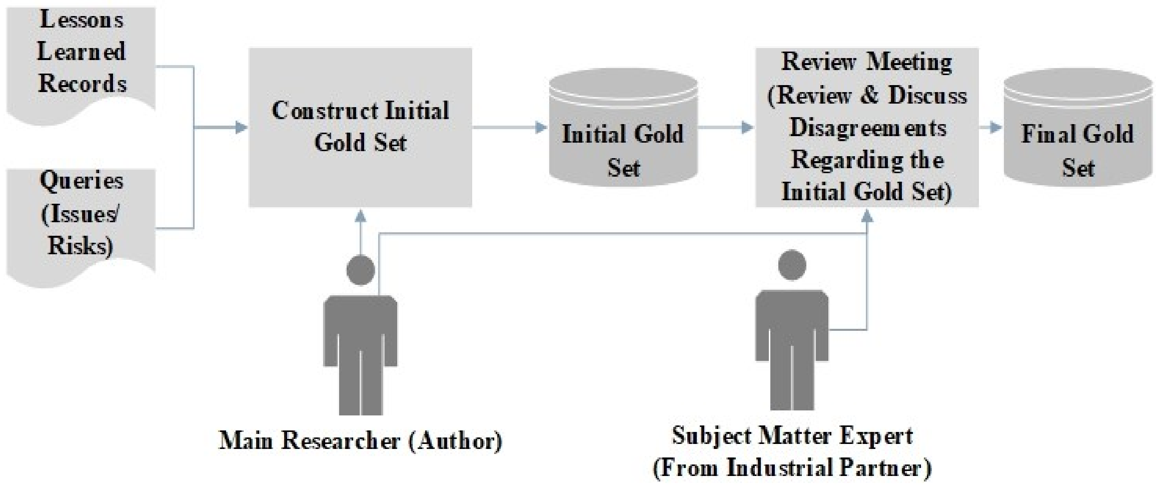
Lessons Learned Records
Queries (Issues/ Risks)
Construct Initial Gold Set
Initial Gold Set
Review Meeting (Review & Discuss Disagreements Regarding the Initial Gold Set)
Final Gold Set
Main Researcher (Author)
Subject Matter Expert (From Industrial Partner)

**Figure 5-1 Gold Set Construction Process**

## 5.2 LL Classifiers Construction

### 5.2.1 Data Preprocessing Tool

As described in Chapter 4 (Section 4.3.1.2), we employed two of the popular preprocessing steps from the natural language processing (NLP), namely *stemming* and *stopping*, to perform the dataset preprocessing. As aforementioned, we applied these two steps in four combinations: apply none of the steps, apply stemming only, apply stopping only, and apply both stemming and stopping. We used *lscp (version 0.01)* [51] for this purpose. *lscp* is an open source lightweight text preprocessor which was originally developed by S. W. Thomas [51] for source code preprocessing, but can also be used for other text input, such as the LL documents in this case study. The tool is implemented using Perl programming language and can accept multiple preprocessing configuration parameters. For the stemming, *lscp* uses the *Lingua::Stem* Perl's module which employs the Porter's stemming algorithm [60]. We describe the parameters that we used in Table 5-1.

**Table 5-1 lscp Tool Parameters**

| Parameter | Description | Default Value |
|---|---|---|
| inPath | The input files directory | "./in" |
| outPath | The output directory to store the preprocessed files | "./out" |
| numberOfThreads | number of parallel processing threads to employ | 1 |
| isCode | Set as 1 if the input files contain source code, set as 0 if the input files are regular files (as in our case study) | 1 |
| doStemming | Set as 1 to perform stemming, set as 0 for no stemming | 0 |
| doStopwordsEnglish | Set as 1 to perform stopping, set as 0 for no stopping | 0 |

An example of the preprocessing script in the case of applying both stemming and stopping steps is shown in Figure 5-2.

```
use lscp;

my $preprocessor = lscp->new;

$preprocessor->setOption("inPath", "input_test_path");
$preprocessor->setOption("outPath", "output_test_path");
$preprocessor->setOption("isCode", 0);
$preprocessor->setOption("doStemming", 1);
$preprocessor->setOption("doStopwordsEnglish", 1);
```

**Figure 5-2 lscp Preprocessing Example**

### 5.2.2 LDA Classifiers Construction Tool

In this case study, there were two steps involved in retrieving the relevant LL records. First, we constructed the IR classifiers, i.e., the Latent Dirichlet Allocation (LDA) classifiers, given the LL corpus. Second, we indexed and searched the IR classifiers constructed for the records relevant to the queries at hand.

Therefore, we employed the MALLET [53] tool for the construction of the LDA classifiers. MALLET is a popular natural language processing (NLP) tool [41]. The tool is implemented using the Java programming language and provides multiple applications including IR and topic modeling. The constructed LDA classifiers consist of multiple membership and mapping files including the list of terms in the corpus, the word-topic membership and the file-topic membership. In order to construct the classifiers, the input files should be imported to the MALLET tool first. Then, the tool is used to train the LDA classifiers based on the input corpus provided. Also, it is worth mentioning that the MALLET tool automatically optimizes many of the LDA parameters such as *sampling iterations number*, *topic-word smoothing*, *document-topic smoothing*, and *similarity.* We summarize the MALLET main configuration parameters used to train the LDA classifiers in Table 5-2.

**Table 5-2 MALLET Tool Parameters**

| Parameter | Description |
|---|---|
| --input | The imported data path |
| --num-topics | number of topics to be created by the LDA model |
| --output-topic-keys | file path to create a map of the words in each topic |
| --output-doc-topics | file path to create a document-topic membership |
| --topic-word-weights-file | file path to create a word-topic membership |

After constructing the LDA classifiers, we used another tool called *lucene-lda* [54] for classifiers' indexing in preparation for searching these classifiers for the queries. To achieve this, we inputted the MALLET generated membership— after applying some formatting to make it suitable for the *lucene-lda* tool— to the *lucene-lda* tool for indexing. After indexing, the tool was ready to execute the queries at hand and return the LL list.

The *lucene-lda* is an open source tool developed by S.W.Thomas [54] using Java programming language. The tool employed the well-known *Apache lucene* [61] open source indexing and searching tool for the indexing of the LDA models topics and topic memberships. For the indexing process, the tool generates a helper class called *LDAHelper()* to store some of the automatically generated configuration parameters while constructing the LDA classifier by the MALLET tool [53]. Moreover, it is important to highlight that the *lucene-lda* tool employs the *conditional probability* method for *similarity*. We summarize the parameters used for indexing and searching in Table 5-3.

**Table 5-3 lucene-lda Tool Parameters (LDA mode)**

| Indexing Parameters | |
|---|---|
| **Parameter** | **Description** |
| <inDir> | Directory path of the original preprocessed documents (LL records) |
| <outIndexDir> | Directory path to generate the indexing data |
| <outLDAIndex> | Path to generate a helper LDA indexing class (LDAHelper()) |
| fileCodes | Path of a file that contains the original names of the preprocessed files |
| ldaConfig | Expects two-part value:<br>1-The number of topics used while constructing the LDA classifier<br>2- The directory path of the associated mapping and topic membership files generated by the MALLET tool (after reformatting) |

| Searching Parameters | |
|---|---|
| **Parameter** | **Description** |
| <indexDir> | Directory path of the generated indexing data for the LDA classifier to be used |
| <LDAIndexDir> | Path of the generated helper LDA indexing class (LDAHelper()) |
| <queryDir> | Directory path of the query documents to be executed |
| <resultsDir> | Directory path to export the retrieved list corresponding to each of the executed queries |
| K | Number of topics configured while constructing the LDA classifier |

### 5.2.3 VSM Classifiers Construction Tool

For the Vector Space Model (VSM) classifiers construction and search, the same *lucene-lda* tool, which was used for LDA, was employed but with the VSM query mode. The same indexing process that was used for LDA was used for VSM, except that there was no need for the generated topic membership files from MALLET, since the indexing was totally handled by the integrated *lucene* tool. Since VSM is the default IR model used by *lucene,* the searching or query execution was mainly handled by the integrated *lucene* tool. A summary of the configured *lucene-lda* tool's parameters, used in indexing and searching in the VSM mode, is shown in Table 5-4.

**Table 5-4 lucene-lda Tool Parameters (VSM mode)**

| **Indexing Parameters** | |
|---|---|
| **Parameter** | **Description** |
| <inDir> | Directory path of the original preprocessed documents (LL records) files |
| <indexDir> | Directory path to generate the indexing data |
| fileCodes | Path of a file contains the original names of the preprocessed files |
| **Searching Parameters** | |
| **Parameter** | **Description** |
| <indexDir> | Directory path of the generated indexing data |
| <queryDir> | Directory path of the query documents to be executed |
| <resultsDir> | Directory path to export the retrieved list corresponding to each of the executed queries |
| weightingCode | term weighting method: 1=tf-idf, 2=Sublinear tf-idf, 3=Boolean |
| scoringCode | similarity scoring method: 1=Cosine, 2=Overlap |

### 5.2.4 LSI Classifiers Construction Tool

In the case of the Latent Semantic Indexing (LSI) classifiers, we employed the *gensim* open source tool [62]. *gensim* is a topic modeling tool which is implemented in Python programming language. Regarding the indexing and searching of the LSI classifiers, we employed the document similarity server gensim's library (*simserver*). The *simserver* library has the ability to construct the LSI classifiers given the input LL documents, index the trained classifiers for future searching, and execute the queries to retrieve the relevant records.

It is important to highlight that *cosine similarity* is the only similarity method implemented by default in *genism* regarding the LSI modeling. In addition, regarding the term weight method, *genism* calculates the term weight by multiplying the returned values of both the local component method *wlocal* and the global component method *wglobal*. For example, in the case of the *tf-idf* term weight method, the *wlocal* is used to calculate the term frequency part (*tf*), where the *wglobal* is used to calculate the document inverse frequency part (*idf*). So, in order to implement all of the term weight methods considered for this study, we had to customize the corresponding *wlocal* and *wglobal* for each of these term weight methods. Then, we inputted the customized methods to the *gensem* server to use it for the LSI classifier training. A summary of the *simserver*'s configured parameters is shown in Table 5-5.

**Table 5-5 simserver Tool Parameters**

| **Training Parameters** | |
|---|---|
| **Parameter** | **Description** |
| method | method to be used to train the topic model (we set it as 'lsi') |
| wlocal | method to calculate the term weight local component |
| wglobal | method to calculate the term weight global component |
| topics | the number of topics remaining after the SVD reduction |
| **Searching Parameters** | |
| **Parameter** | **Description** |
| doc | pointer to the query id to be executed to search using the loaded LSI model |
| min_score | minimum similarity score for a retrieved document within retrieved list |
| max_results | maximum number of records to be retrieved within the retrieved list |

## 5.3 Results

In this section, we describe the results of all the LL classifiers considered. For the evaluation of all the considered LL classifiers, we followed the evaluation process defined in Chapter 4 (Section 4.4). To make it easy to follow, we discuss the results based on each of the chosen performance metrics, top-K and MAP, separately in the following two subsections. Each subsection starts with the overall discussion of the performance results, and then it demonstrates the statistical test results regarding the significant effect of the parameter configurations on the classifier results. Also, we share the results of all classifiers in Appendices B and C as a reference for interested practitioners and researchers.

### 5.3.1 Top-K Results

The top-20, K is set to 20 (refer to Chapter 4, Section 4.4.1, for details), performance results regarding the best four classifiers and the worst four classifiers for each of the IR models considered, VSM, LSI, and LDA, are illustrated in Table 5-6. When observing the highest performing classifier in each technique, the best top-20 results of 70% are recorded by the VSM and LSI top two classifiers, while the lowest performance is recorded by the LDA top classifier with only 52%. So, the top VSM and LSI classifiers outperformed the top LDA classifier, which is consistent with the literature results for similar problems, such as bug localization [42][44]. An observation regarding the best two classifiers of VSM and LSI is that both classifiers miss the relevant LL records for almost the same queries (issues/risks) except for only one query. All these queries have only three or fewer relevant LL records, which makes them hard queries, except for only one query which has seven relevant LL records according to the gold set. This indicates that the VSM and LSI best classifiers can be considered good retrieval classifiers for the evaluation dataset at hand.

**Table 5-6 Lessons Learned Classifiers Top-K Performance Results (Best Four and Worst Four Classifiers)**

| **VSM** | | | **LSI** | | | **LDA** | | |
|---|---|---|---|---|---|---|---|---|
| Rank | Parameters Values | Top-20 (%) | Rank | Parameters Values | Top-20 (%) | Rank | Parameters Values | Top-20 (%) |
| 1 | Stemming+tf-idf+cosine | 70 | 1 | None+tf-idf+cosine +128 topics | 70 | 1 | Stemming and stopping +32 topic | 52 |
| 2 | Stemming+sublinear tf-idf+ cosine | 69 | 2 | None+sublinear tf-idf +cosine+128 topic | 69 | 2 | Stopping+32 topic | 46 |
| 3 | None+sublinear tf-idf+ cosine | 61 | 3 | Stemming+sublinear tf-idf+cosine+256 topic | 69 | 3 | Stemming and stopping +64 topic | 46 |
| 4 | Stemming and stopping+ sublinear tf-idf+cosine | 61 | 4 | None+tf-idf+ cosine+256 topic | 69 | 4 | None+32 topic | 41 |
| 21 | Stemming+tf-idf+overlap | 52 | 45 | Stemming+boolean +cosine+64 topic | 50 | 13 | Stemming and stopping +128 topic | 26 |
| 22 | Stemming+boolean+ overlap | 50 | 46 | None+boolean+ cosine+128 topic | 48 | 14 | Stemming+ 256 topic | 22 |
| 23 | None+boolean+ cosine | 46 | 47 | None+boolean+ cosine+64 topic | 44 | 15 | None+256 topic | 19 |
| 24 | None+boolean+ overlap | 46 | 48 | None+boolean+ cosine+32 topic | 43 | 16 | Stemming and stopping +256 topic | 19 |

In addition, the descriptive statistics of the top-20 performance results, in Table 5-7, demonstrate that the parameter configurations of the LL classifiers have a significant effect on the results. In the case of the VSM classifiers, there is a significant difference, about 50% relative improvement (calculated as $\frac{70-46}{46}$%), in the performance between the best classifier, 70%, and the worst classifier, 46%, and this can also be observed between the best classifier, 70%, and the median classifier, 54%. The same observation is true for the LSI and LDA classifiers, as depicted in Table 5-7.

**Table 5-7 Top-K Descriptive Statistics**

| **VSM** | | **LSI** | | **LDA** | |
|---|---|---|---|---|---|
| | Top-20 (%) | | Top-20 (%) | | Top-20 (%) |
| Minimum | 46 | Minimum | 43 | Minimum | 19 |
| 1st Quartile | 52 | 1st Quartile | 55 | 1st Quartile | 26 |
| Mean | 56 | Mean | 59 | Mean | 33 |
| Median | 54 | Median | 59 | Median | 35 |
| Standard deviation | 6 | Standard deviation | 7 | Standard deviation | 10 |
| 3rd Quartile | 58 | 3rd Quartile | 65 | 3rd Quartile | 41 |
| Maximum | 70 | Maximum | 70 | Maximum | 52 |

In order to study the impact of the configuration values on the performance results statistically, we apply the Tukey's HSD statistical test to the performance results of each of the parameter configuration values. The results of the Tukey's test, regarding the top-20 performance results, illustrated in Table 5-8 at a confidence level of 95%, are demonstrated in the following two subsections, in which we use the short term "*performance results*" to refer to the top-20 performance results. Also, the results at 90% and 99% confidence levels are shared in Appendix D.

**Table 5-8 Tukey's HSD Statistical Test Results (Top-K) (95% Confidence Level)**

| VSM | | | LSI | | | LDA | | |
|---|---|---|---|---|---|---|---|---|
| Group | Mean (%) | Preprocessing steps | Group | Mean (%) | Preprocessing steps | Group | Mean (%) | Preprocessing steps |
| A | 59 | Stemming and stopping | A | 60 | Stopping | A | 36 | Stemming and stopping |
| A | 58 | Stemming | A | 60 | Stemming and stopping | A | 34 | Stopping |
| A | 53 | None | A | 60 | Stemming | A | 32 | None |
| A | 53 | Stopping | A | 58 | None | A | 32 | Stemming |
| Group | Mean (%) | Similarity | Group | Mean (%) | Number of topics | Group | Mean (%) | Number of topics |
| A | 58 | Cosine | A | 63 | 128 | A | 45 | 32 |
| B | 53 | Overlap | A | 61 | 256 | AB | 37 | 64 |
| | | | AB | 60 | 64 | B | 28 | 128 |
| | | | B | 54 | 32 | B | 24 | 256 |
| Group | Mean (%) | Term weight | Group | Mean (%) | Term weight | | | |
| A | 58 | tf-idf | A | 63 | tf-idf | | | |
| A | 57 | Sublinear tf-idf | A | 63 | Sublinear tf-idf | | | |
| A | 52 | Boolean | B | 53 | Boolean | | | |

#### 5.3.1.1 Lessons Learned Classifier Parameters Statistical Test Results

Regarding the similarity method, in the VSM case, the HSD test results show a significant difference in the performance results when using the *cosine* method versus the results of using the *overlap* method (See Figure 5-3). This means that the similarity method employed has an impact on the performance results for the dataset considered in this case study. The *cosine* similarity method shows the best performance results and comes in the top group. On the other hand, the *overlap* method results come in the bottom group.

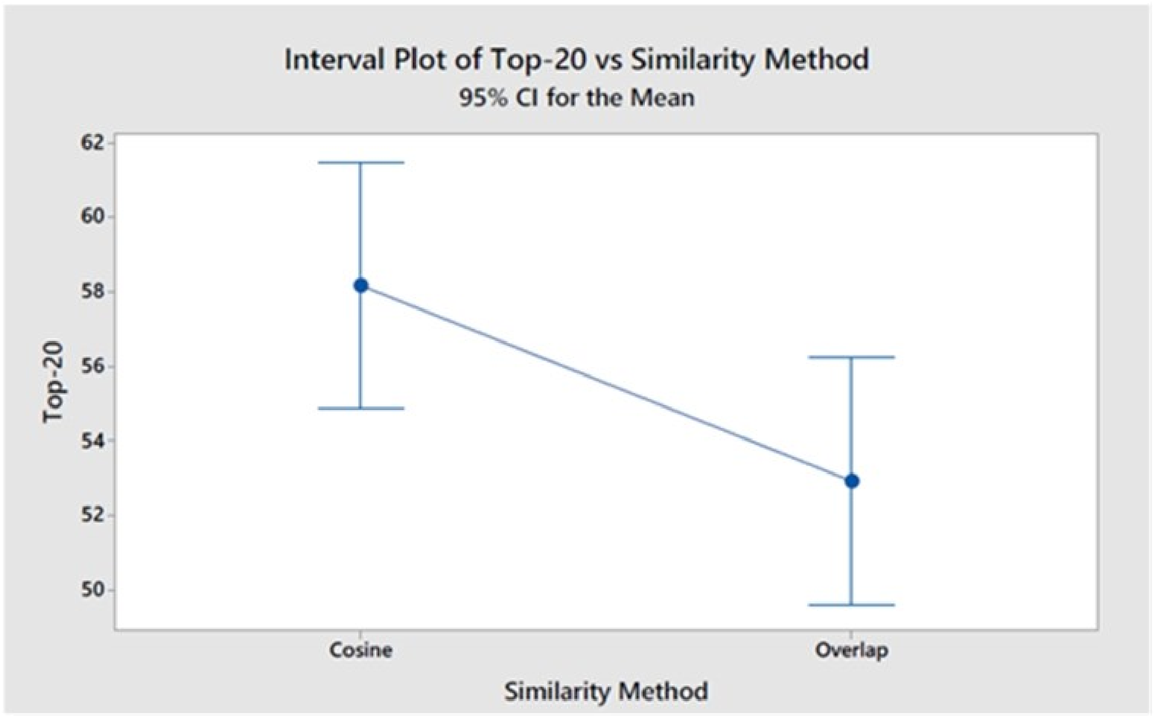

**Figure 5-3 Top-20 Statistical Test Results for VSM (Similarity Methods)**

Regarding the term weight, in the VSM case, the test results show that there is no statistically significant difference when changing the parameter value between the *tf-idf*, *sublinear tf-idf* and *Boolean* weighting methods (See Figure 5-4).

For the LSI classifiers, the statistical test shows that the term weight parameter has a statistically significant impact on the performance results. Both the *tf-idf* and *sublinear tf-idf* weighting methods come in the top group and have the highest top-20 performance results, while the *Boolean* weighting method comes in the bottom group with the lowest performance results (See Figure 5-5).

An overall observation, regarding the *term weight* parameter, is that the *tf-idf* weighting method always shows the highest performance results for both the VSM and LSI models, followed by the *sublinear tf-idf* method, although there is no statistical significance for VSM as described, which is consistent with the results from other IR application studies [42].

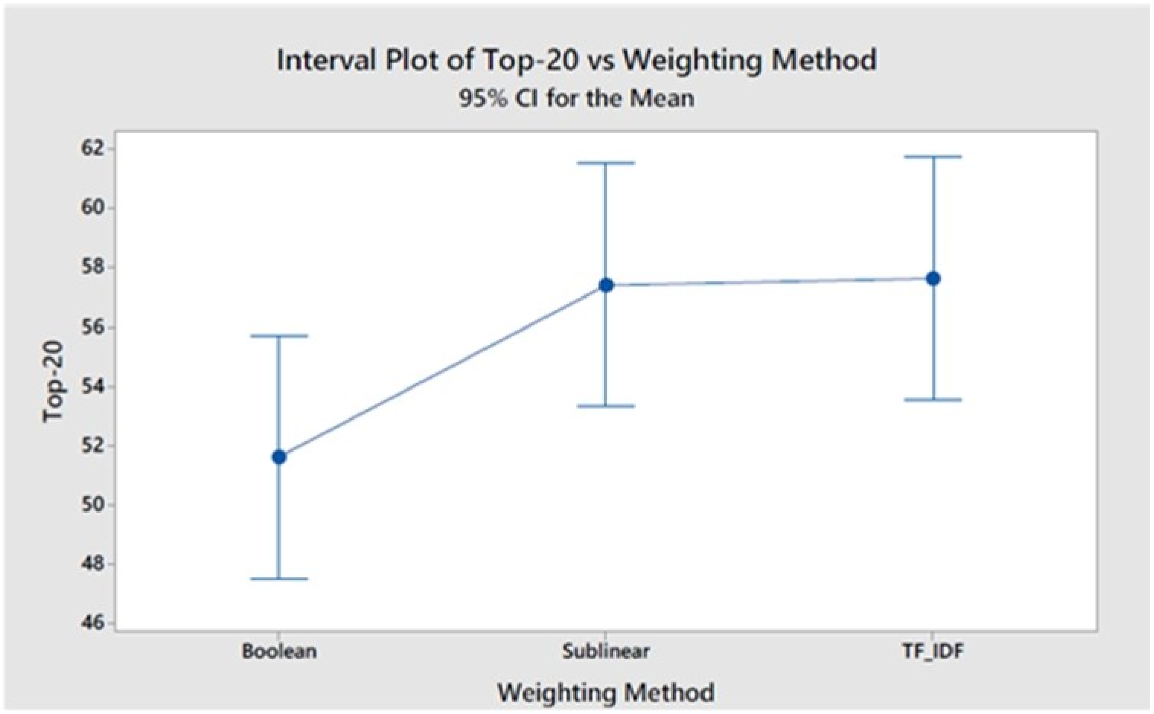

**Figure 5-4 Top-20 Statistical Test Results for VSM (Term Weighting Methods)**

Regarding the *number of topics,* the HSD test has revealed that it has a statistically significant impact on the classifiers' performance results. This means that the performance results differ when the classifiers are configured with different topic numbers. This applies for both the LSI and LDA classifiers. However, for LSI, the largest numbers of topics, "128" and "256," come in the top group. This indicates that the more topics used, the better the performance results. On the other hand, for the LDA classifiers, the situation is different, where the smallest numbers of topics, "32" and "64," come in the top groups (See Figure 5-6 and Figure 5-7).

#### 5.3.1.2 Preprocessing Steps Statistical Test Results

Table 5-8 illustrates the HSD test results of applying the four preprocessing combinations on the classifiers' top-20 performance, where there is no statistically significant difference in the results when applying any of the preprocessing steps. This is the case for all the IR models considered, VSM, LSI, and LDA, within the context of the dataset at hand. However, applying both *stemming* and *stopping* together, in the case of VSM and

LDA, and applying only *stopping*, in the case of LSI, give the highest top-20 performance (See Figure 5-8, Figure 5-9, and Figure 5-10).

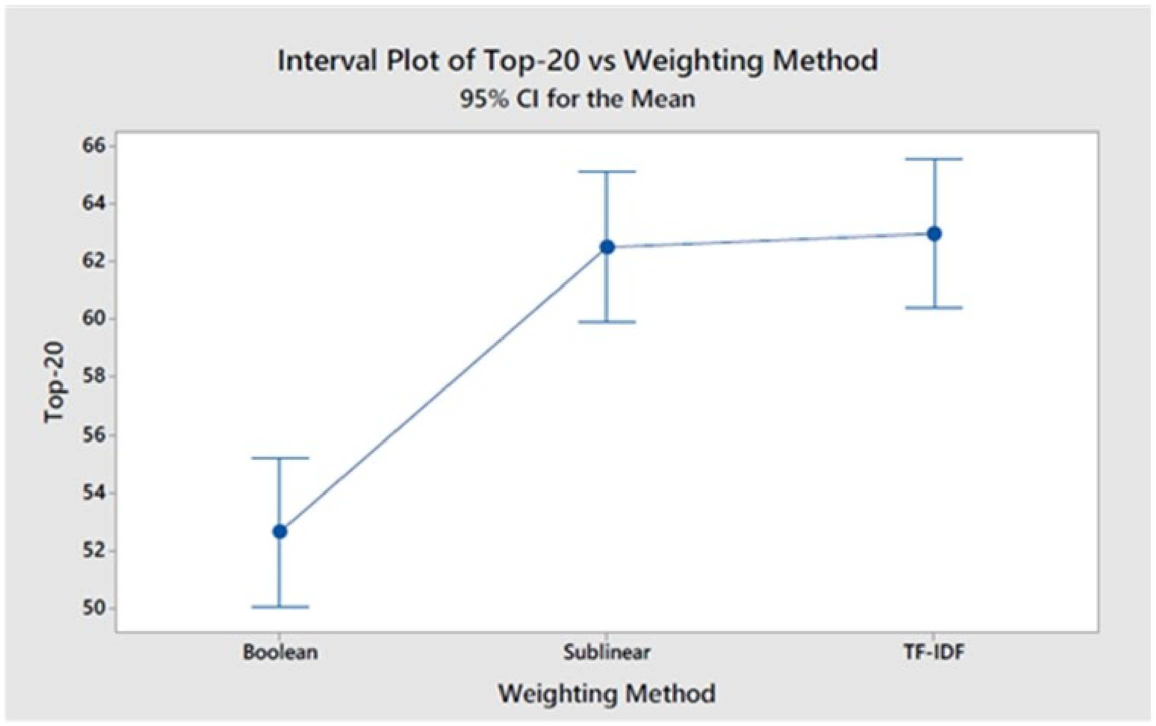

**Figure 5-5 Top-20 Statistical Test Results for LSI (Term Weighting Methods)**

### 5.3.2 MAP Results

Table 5-9 lists the MAP performance results regarding the best four classifiers and the worst four classifiers for each of the IR models considered.

After analyzing the MAP results, we conclude that some of the insights from the top-20 results still apply. When looking at the top performing classifiers in each model, the highest MAP result of 0.198 is recorded by the top classifier in LSI, followed by 0.189 in VSM, which is similar to the top-20 results. These MAP performance results are satisfactory compared to other studies from the literature [44][63]. Also, as in the top-20 results, the LDA top classifier achieves the lowest performance of 0.096, compared to the top performing classifiers in VSM and LSI. In addition, the worst results for both the

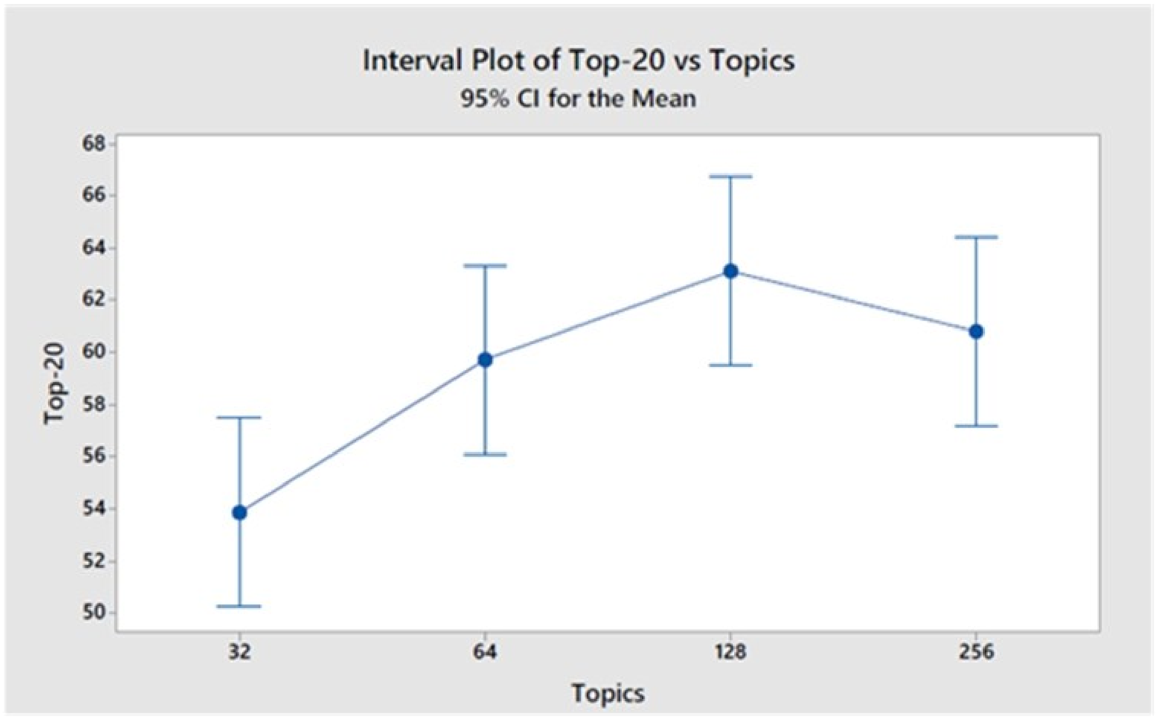

**Figure 5-6 Top-20 Statistical Test Results for LSI (Number of Topics)**

VSM and LSI classifiers, 0.081 and 0.085, respectively, are slightly different from the LDA top classifier result of 0.096. So, again, the MAP results are aligned with both of the top-20 results, from this case study, and the literature results, which provide evidence of the superiority of both VSM and LSI classifier results over LDA classifiers in different empirical studies [42][44].

Similar to the top-20 results, the descriptive analysis of the MAP performance results, presented in Table 5-10, indicates that the classifier configuration has a remarkable impact on the performance. This can be inferred from the difference between the VSM best classifier performance of 0.189 and the VSM worst classifier performance of 0.081, which represents more than 100% relative improvement. Also, there is a high difference between the median VSM classifier, 0.122, and the minimum VSM classifier. The same insight applies for both the LSI and LDA results.

**Table 5-9 Lessons Learned Classifiers MAP Performance Results (Best Four and Worst Four Classifiers)**

| VSM | | | LSI | | | LDA | | |
|---|---|---|---|---|---|---|---|---|
| Rank | Parameter values | MAP | Rank | Parameter values | MAP | Rank | Parameter values | MAP |
| 1 | Stemming and stopping+ sublinear tf-idf+cosine | 0.189 | 1 | Stemming and stopping+ sublinear tf-idf+cosine+ 128 topic | 0.198 | 1 | Stemming +32 topic | 0.096 |
| 2 | Stemming and stopping+ tf-idf+cosine | 0.188 | 2 | Stemming and stopping+ tf-idf+ cosine+128 topic | 0.198 | 2 | Stemming and stopping+ 32 topic | 0.089 |
| 3 | Stemming+tf-idf+cosine | 0.156 | 3 | Stopping+tf-idf+ cosine+64 topic | 0.194 | 3 | None+32 topic | 0.082 |
| 4 | Stemming+ sublinear tf-idf+cosine | 0.153 | 4 | Stopping+ sublinear tf-idf+cosine+ 64 topic | 0.194 | 4 | Stopping+ 32 topic | 0.075 |
| 21 | None+tf-idf+overlap | 0.099 | 45 | None+ boolean+ cosine+ 128 topic | 0.107 | 13 | Stemming and stopping+ 128 topic | 0.040 |
| 22 | None+sublinear tf-idf+ overlap | 0.095 | 46 | None+ boolean+ cosine+ 64 topic | 0.096 | 14 | Stopping+ 64 topic | 0.036 |
| 23 | None+boolean+ cosine | 0.082 | 47 | Stemming+ boolean+ cosine+32 topic | 0.086 | 15 | None+256 topic | 0.031 |
| 24 | None+boolean+ overlap | 0.081 | 48 | None+ boolean+ cosine+ 32 topic | 0.085 | 16 | Stemming and stopping+ 256 topic | 0.030 |

In the following subsections, we demonstrate the HSD statistical test results, at 95% confidence level, listed in Table 5-11, regarding the significant effect of the LL classifiers' configuration on the MAP performance results. We refer to the MAP performance results as "*performance results*" in the following two subsections. Also, the results at 90% and 99% confidence levels are shared in Appendix D.

**Table 5-10 MAP Descriptive Statistics**

| VSM | | LSI | | LDA | |
|---|---|---|---|---|---|
| | MAP | | MAP | | MAP |
| Minimum | 0.081 | Minimum | 0.085 | Minimum | 0.030 |
| 1st Quartile | 0.111 | 1st Quartile | 0.132 | 1st Quartile | 0.043 |
| Mean | 0.126 | Mean | 0.153 | Mean | 0.058 |
| Median | 0.122 | Median | 0.163 | Median | 0.057 |
| Standard deviation | 0.028 | Standard deviation | 0.029 | Standard deviation | 0.020 |
| 3rd Quartile | 0.142 | 3rd Quartile | 0.172 | 3rd Quartile | 0.065 |
| Maximum | 0.189 | Maximum | 0.198 | Maximum | 0.096 |

**Table 5-11 Tukey's HSD Statistical Test Results (MAP) (95% Confidence Level)**

| VSM | | | LSI | | | LDA | | |
|---|---|---|---|---|---|---|---|---|
| Group | Mean | Preprocessing steps | Group | Mean | Preprocessing steps | Group | Mean | Preprocessing steps |
| A | 0.159 | Stemming and stopping | A | 0.170 | Stemming and stopping | A | 0.067 | Stemming |
| B | 0.126 | Stemming | A | 0.164 | Stopping | A | 0.056 | Stemming and stopping |
| B | 0.117 | Stopping | AB | 0.147 | Stemming | A | 0.055 | None |
| B | 0.102 | None | B | 0.132 | None | A | 0.053 | Stopping |
| Group | Mean | Similarity | Group | Mean | Number of topics | Group | Mean | Number of topics |
| A | 0.135 | Cosine | A | 0.161 | 128 | A | 0.085 | 32 |
| A | 0.117 | Overlap | A | 0.161 | 64 | B | 0.053 | 64 |
| | | | A | 0.158 | 256 | B | 0.051 | 128 |
| | | | A | 0.133 | 32 | B | 0.042 | 256 |
| Group | Mean | Term weight | Group | Mean | Term weight | | | |
| A | 0.136 | tf-idf | A | 0.167 | Sublinear tf-idf | | | |
| A | 0.134 | Sublinear tf-idf | A | 0.166 | tf-idf | | | |
| A | 0.109 | Boolean | B | 0.127 | Boolean | | | |

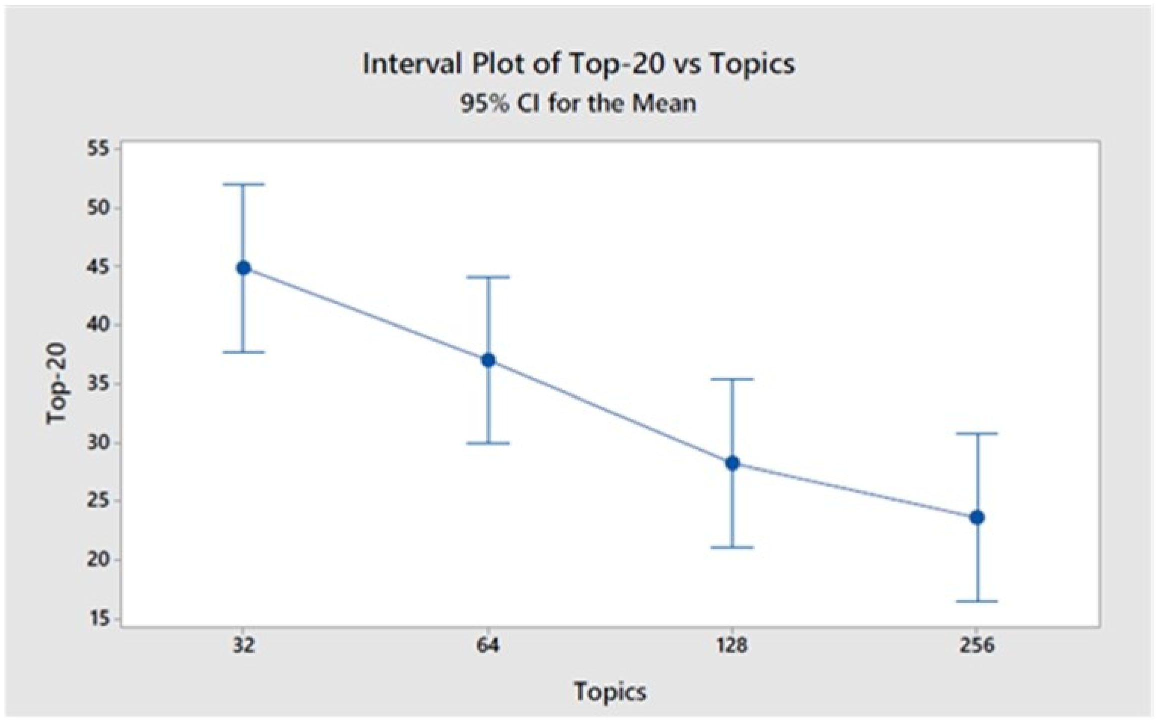

**Figure 5-7 Top-20 Statistical Test Results for LDA (Number of Topics)**

### 5.3.2.1 Lessons Learned Classifier Parameters Statistical Test Results

In the case of the VSM classifiers, the Tukey's test results demonstrate that the classifier parameter values have no significant impact on the performance results. This means that both within the context of this case study dataset and the experiments that were conducted, neither the *similarity* parameter nor the *term weight* parameter affects the performance of the VSM classifiers (See Figure 5-11 and Figure 5-12).

This is not exactly the same for the LSI classifiers, where the statistical test results reveal the significant impact of the *term weight* parameter on the classifier performance results. The *sublinear tf-idf* term weighting method records the highest mean performance value, 0.167, and shares the top group with the *tf-idf* method, while the *Boolean* method comes in the bottom group (See Figure 5-13). On the other hand, the statistical test of the

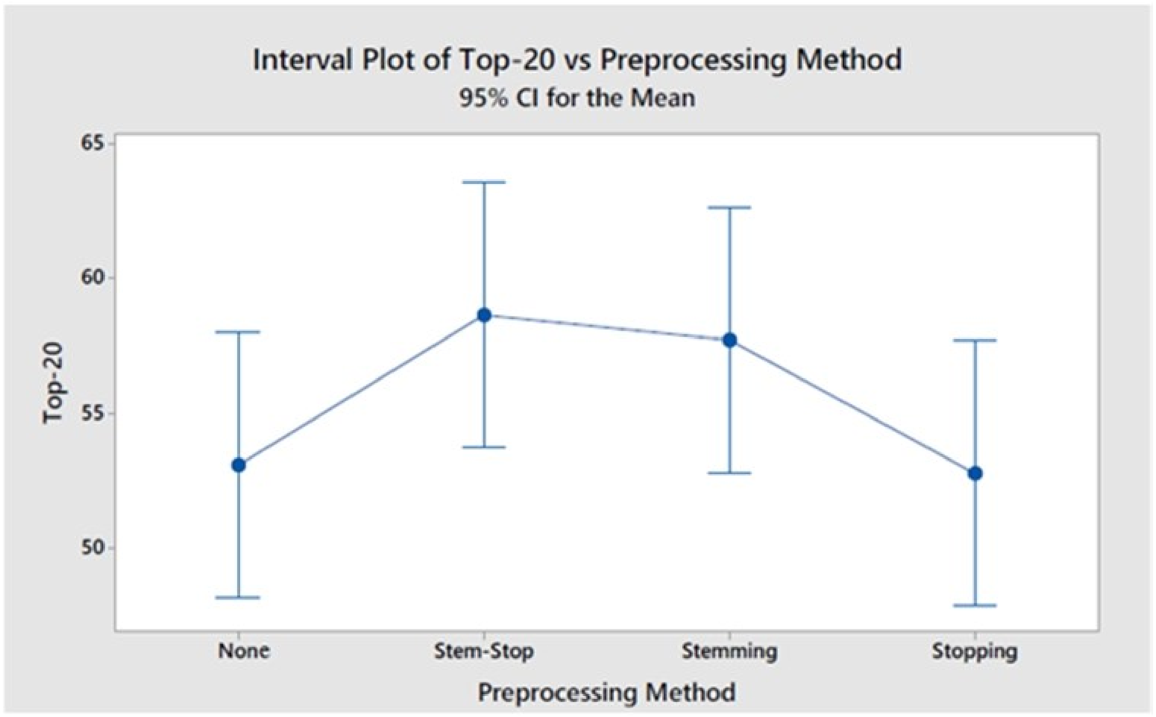

**Figure 5-8 Top-20 Statistical Test Results for VSM (Preprocessing Method)**

*number of topics* parameter demonstrates no significant difference in the performance results (See Figure 5-14).

For the LDA classifiers, a significant difference in the *number of topics* parameter results

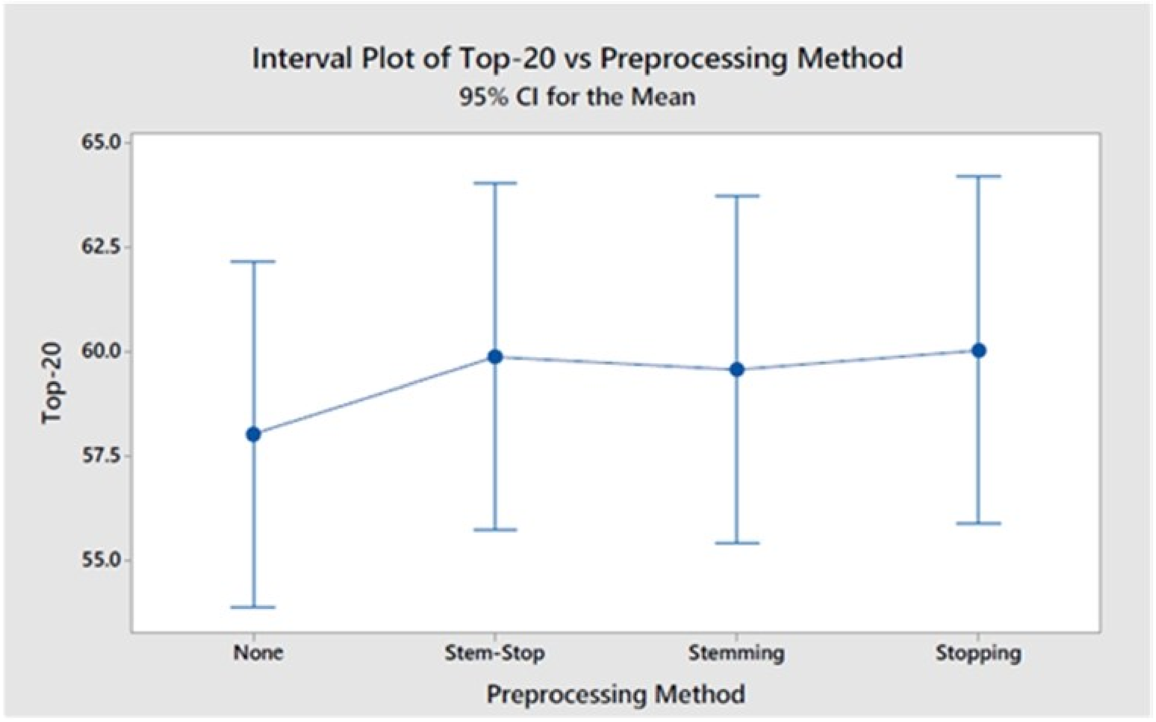

**Figure 5-9 Top-20 Statistical Test Results for LSI (Preprocessing Method)**

is reported by the statistical test. The top group comprises the performance results of the "32" topic classifiers, while the bottom group involves the performance results corresponding to "64," "128" and "256" topic configuration values (See Figure 5-15).

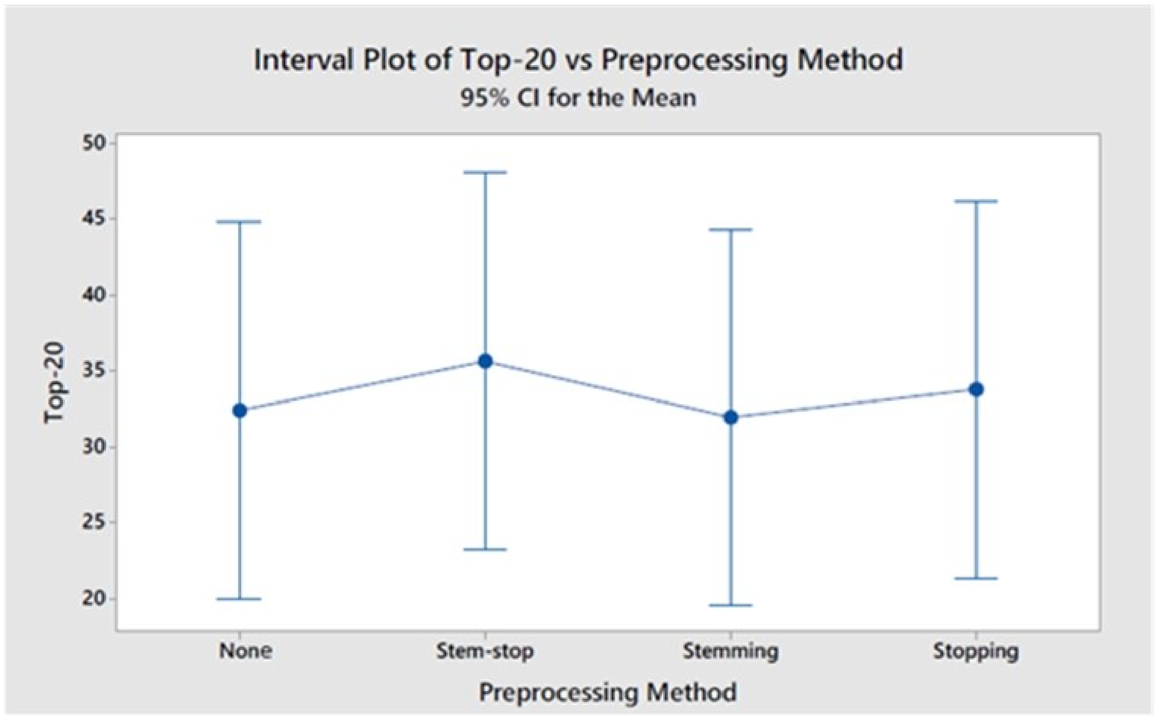

**Figure 5-10 Top-20 Statistical Test Results for LDA (Preprocessing Method)**

#### 5.3.2.2 Preprocessing Steps Statistical Test Results

In the case of the VSM classifiers, the HSD test shows a significant impact from the *preprocessing steps* parameter, where applying both the *stemming* and *stopping* together comes in the top group, while the application of other preprocessing steps, including *stemming* alone, *stopping* alone, and using *none of the preprocessing* steps, comes in the bottom group (See Figure 5-16).

For the LSI classifiers, both *preprocessing steps* configurations of applying the *stemming* and *stopping* steps together, and only the *stopping* step are ranked in the top groups. The *stemming* step is ranked in the middle, and not applying any step comes in the bottom group (See Figure 5-17).

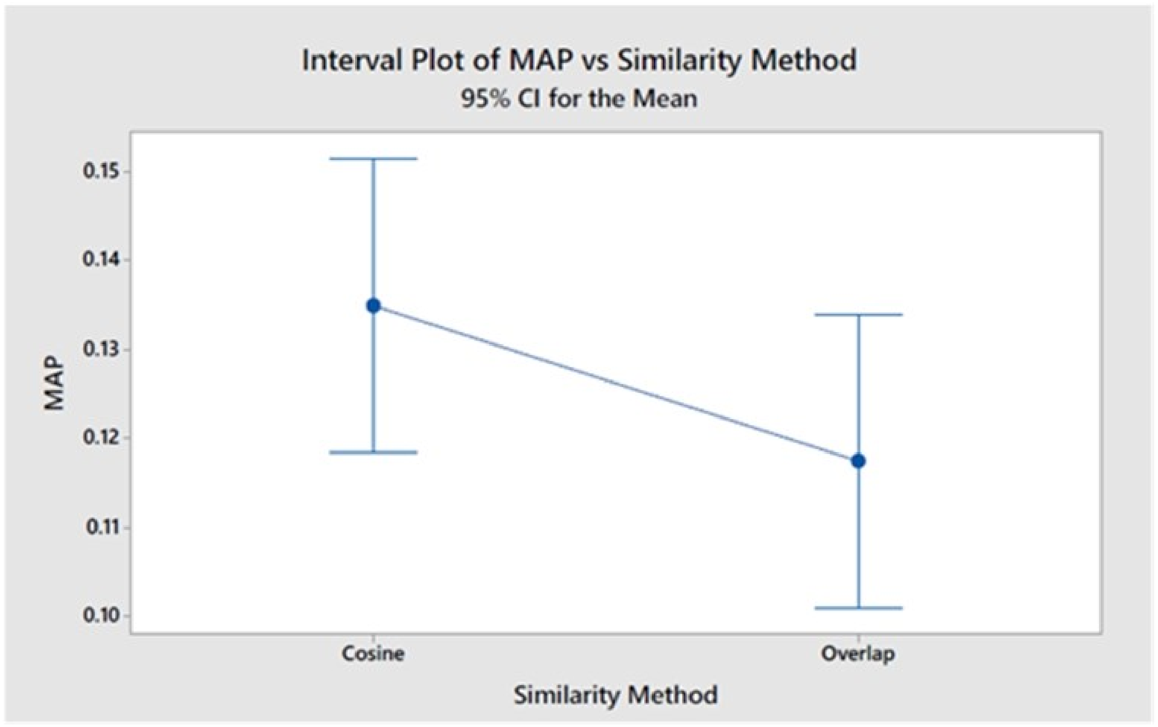

**Figure 5-11 MAP Statistical Test Results for VSM (Similarity Method)**

Regarding the LDA classifiers, the statistical test infers no significant impact for the preprocessing steps on the classifier performance results (See Figure 5-18).

In the following section, we elaborate on the results analysis and provide our overall findings and observations. We then link these findings to the original research questions defined in Chapter 4 (Section 4.2).

## 5.4 Results Discussion

In this section, we provide an overall discussion and demonstrate our overall findings from the results of the case study.

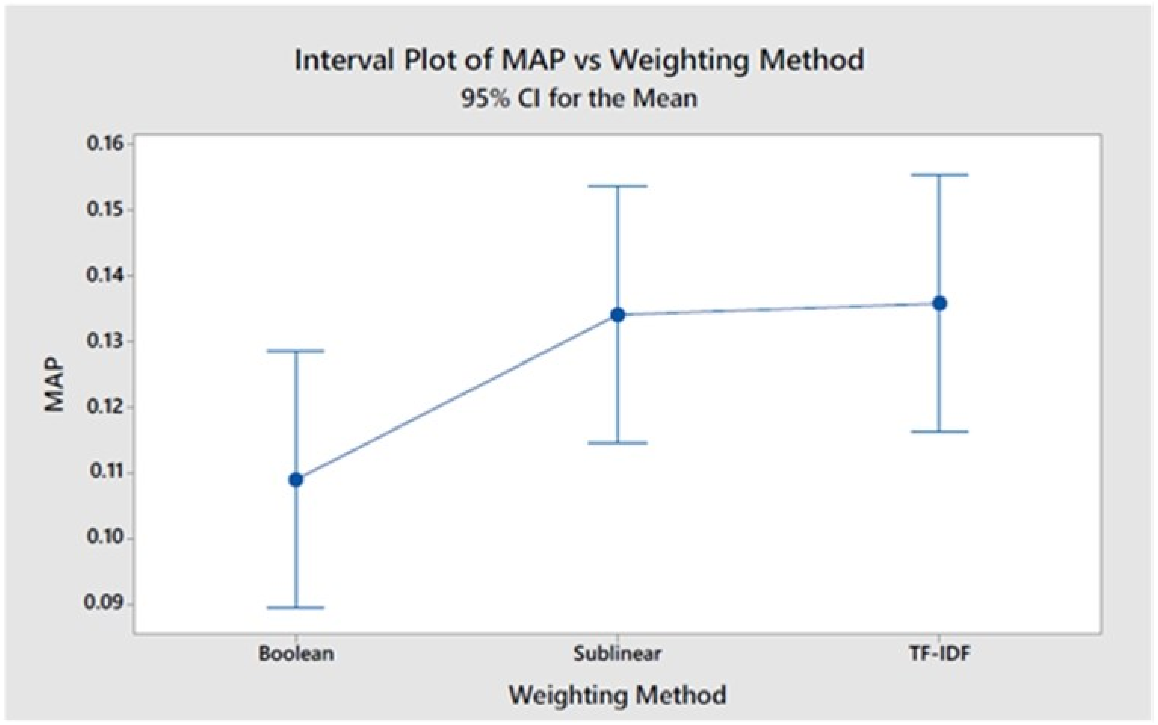

**Figure 5-12 MAP Statistical Test Results for VSM (Term Weighting Method)**

Regarding the research questions, the conclusions are based on the analysis of the performance results of the 88 different LL classifiers considered in the case study. Our conclusions can be summarized as follows:

- Considering the achieved adequate performance results, 70% for top-20 and 0.198 for MAP, we confirm the effectiveness of employing IR techniques in order to automatically push the relevant LL information to PMs within software organizations.

- With this convenient level of performance, practitioners are encouraged to rely on the LL IR-based classifiers to automatically search, within the existing organization’s LL repositories, for relevant solutions regarding their most pressing issues/risks; this answers the first research question *RQ1*.

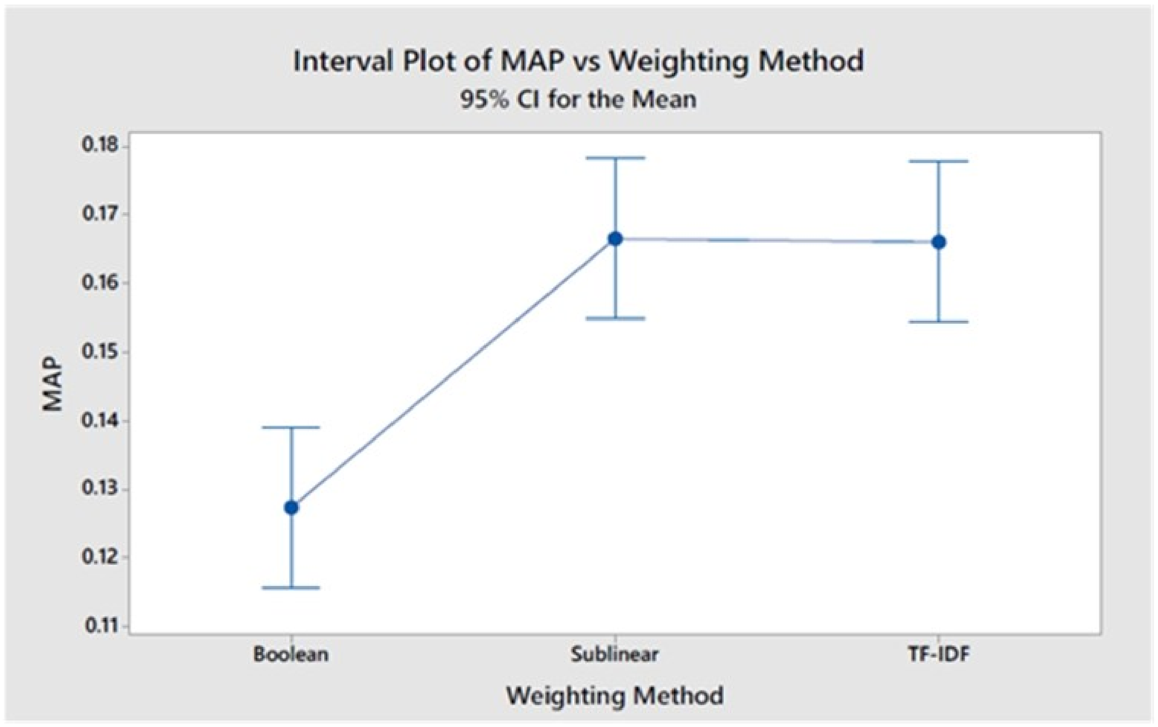

**Figure 5-13 MAP Statistical Test Results for LSI (Term Weighting Method)**

- Relying on the available artifacts, such as project management issues and risk register that are associated with software development and project management processes, to replace the manual querying of the organization's repositories can be significant. This is a positive answer to the second research question *RQ2*, which is supported by the case study results. Since there is no manual querying needed, the practitioners can explore the organization's repositories without worrying about the burden of manually searching the unstructured data, which can be time and effort consuming.

- Regarding the hypothesis of the impact of the classifier configuration on performance, this is generally found to be significant. The same IR technique shows different performance results considering different configurations, and this provides an answer to the third research question, *RQ3*.

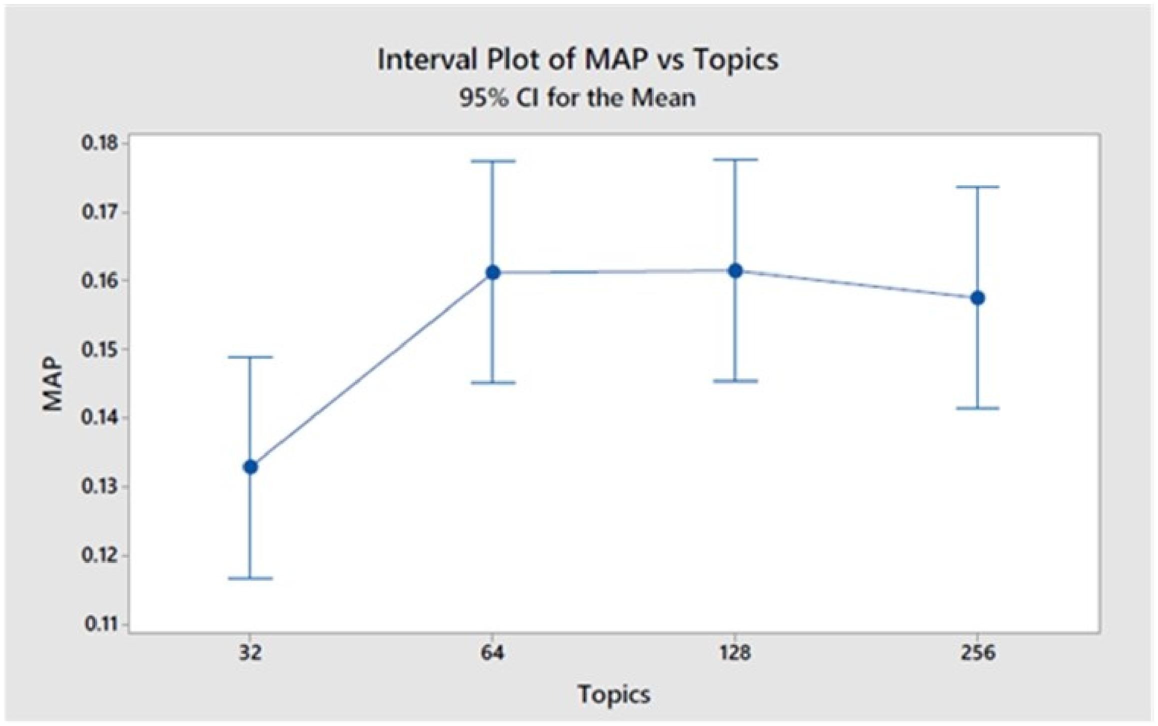

**Figure 5-14 MAP Statistical Test Results for LSI (Number of Topics)**

- In the study, the VSM and LSI IR techniques achieved the best top-20 and MAP performance, followed by LDA.
- Our statistical test of the impact of applying different preprocessing steps shows no significant difference for the top-20 performance results. This can be attributed to our dataset and models. However, since the statistical test of the impact of applying different preprocessing steps shows significance in the MAP results, for VSM and LSI, and in other cases from the literature, such as bug localization [42], we advise considering those different preprocessing steps in future studies.

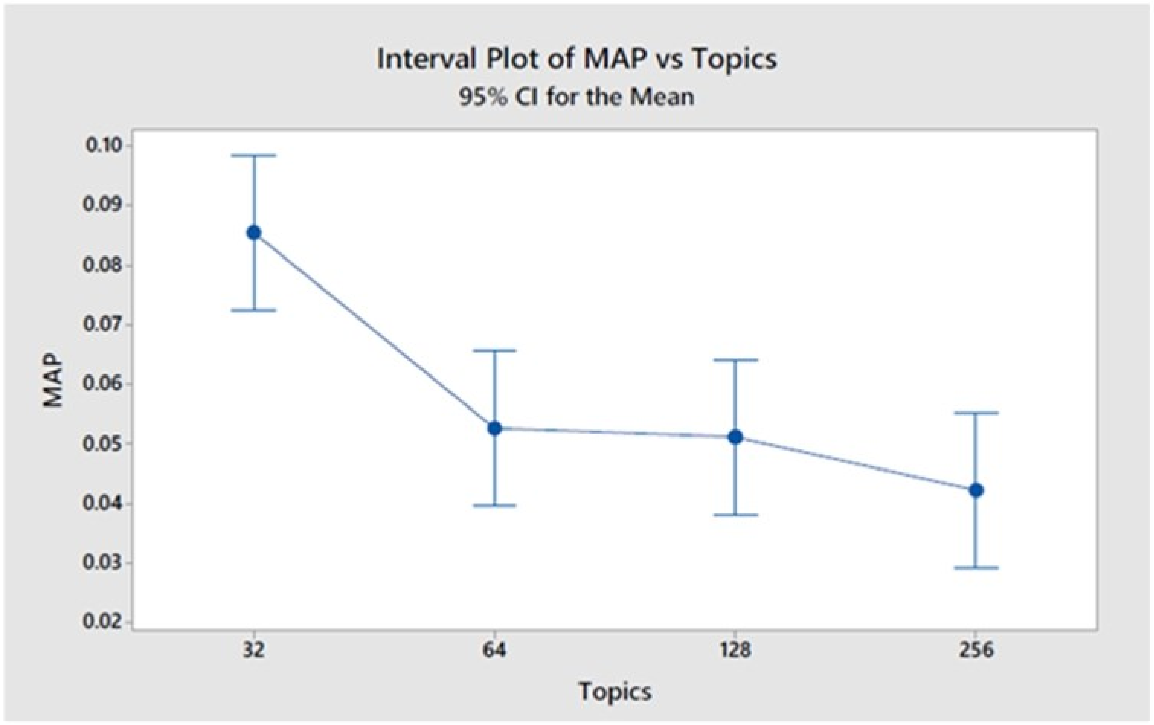

**Figure 5-15 MAP Statistical Test Results for LDA (Number of Topics)**

An overall observation is that the worst VSM and LSI classifier performance results, 46% and 43%, respectively, for the top-20, are slightly lower than the best LDA classifier's performance of 52%. Also, the worst LDA classifier's performance, 19%, is significantly poorer than the worst classifiers in the case of VSM and LSI of 46% and 43%, respectively. The same insight can be inferred from the MAP performance results. This can be considered an indication that the LDA technique is not suitable for the LL recall problem. This indication can be useful for practitioners and researchers who plan to work on similar problems in the future. Also, we advise the consideration of employing the *tf-idf* or *sublinear tf-idf* weighting method together with the *cosine* similarity method, as this combination showed the best classifiers' top-20 and MAP results for both the VSM and LSI techniques.

Since the results indicate that the configurations and the selected IR techniques do matter, we recommend considering different configurations and IR techniques, and to be careful when deciding on the LL classifier to be applied to the problem and dataset in hand.

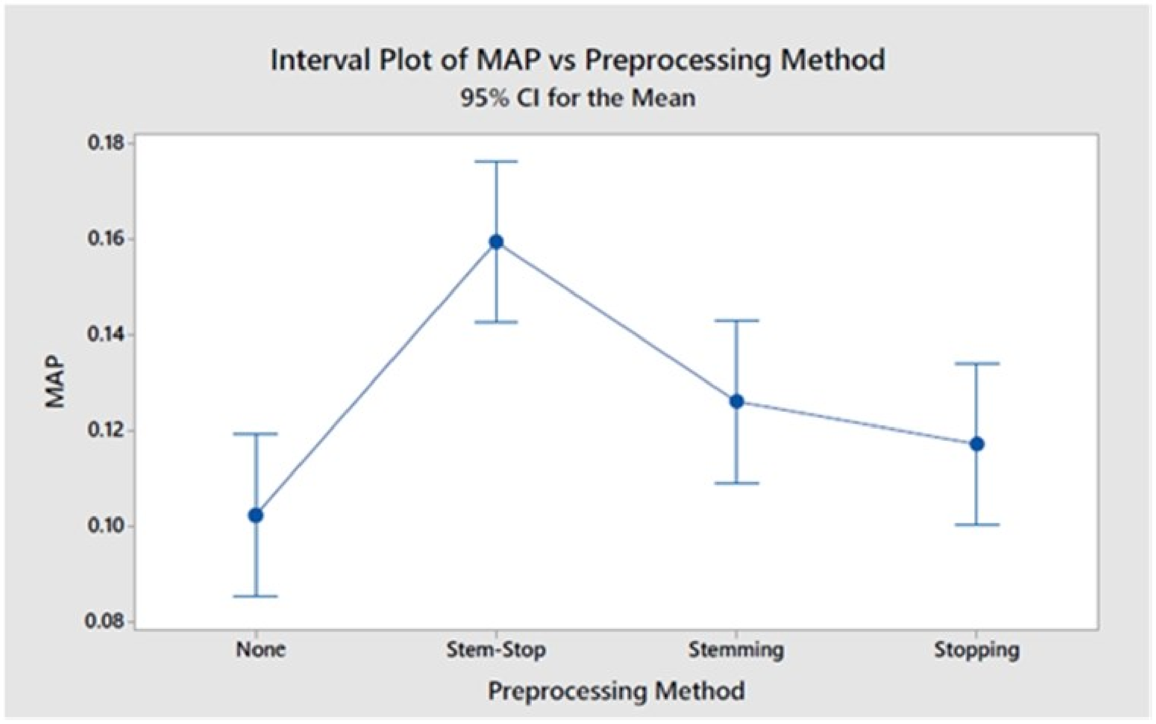

**Figure 5-16 MAP Statistical Test Results for VSM (Preprocessing Method)**

## 5.5 Threats to Validity

In this section, we discuss two validity threats for the empirical study we conducted. These threats involve the gold set, as well as the dataset representation and context.

*Gold set validity*. In this study, we have relied in the classifier validation on the collection constructed of the queries-relevant LL records mapping. As this mapping collection can be subjective and may cause a threat to the validity of the case study and conclusions, we have taken two mitigation steps. First, as a trial to eliminate any bias, we involved two practitioners in the discussion and construction of this mapping collection. Second, after reaching a consensus from the two practitioners regarding this mapping collection, the collection was baselined. So, even if the collection has flaws, such as positive or negative false, the baseline guarantees that the same collection is used to evaluate all the classifiers considered using all the three IR techniques. So, the classifiers were evaluated under the same comparison factors and within the same context.

*Dataset representation.* Although in this empirical study we were keen to consider a significant dataset, including both significant LL records and query records, the dataset considered does not represent all of the LL records in the world or even in the organization. In addition, we were limited to the dataset provided by our industrial partner, which was out of our control because of data confidentiality restrictions. Since this is a common challenge in the context of empirical studies seeking real industrial data, we did our best to come up with solid conclusions by including LL and queries from a variety of projects, domains, and regions. Due to this limitation in the dataset representation, the results and conclusions are not necessarily valid for other contexts. Although our experiment cannot be reproduced, since we cannot share the dataset, due to the non-disclosure agreement limitation, we provide the details of the case study design to encourage researchers and practitioners to proceed with similar methodologies and case studies regarding their different datasets.

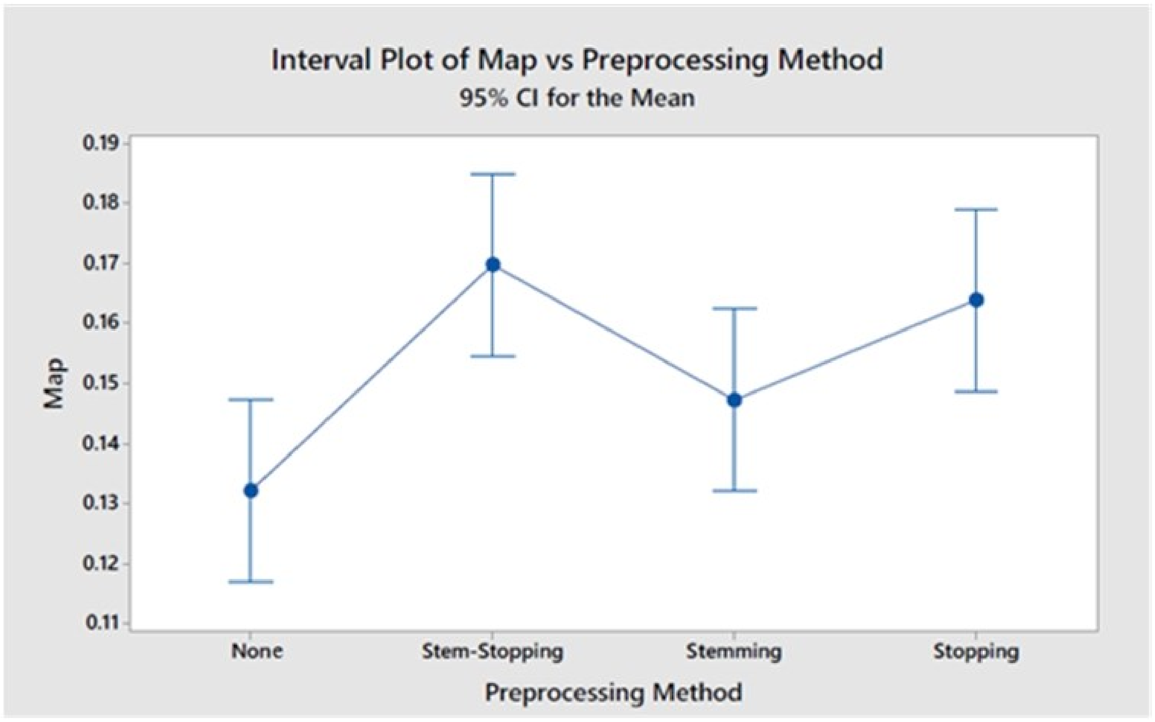

**Figure 5-17 MAP Statistical Test Results for LSI (Preprocessing Method)**

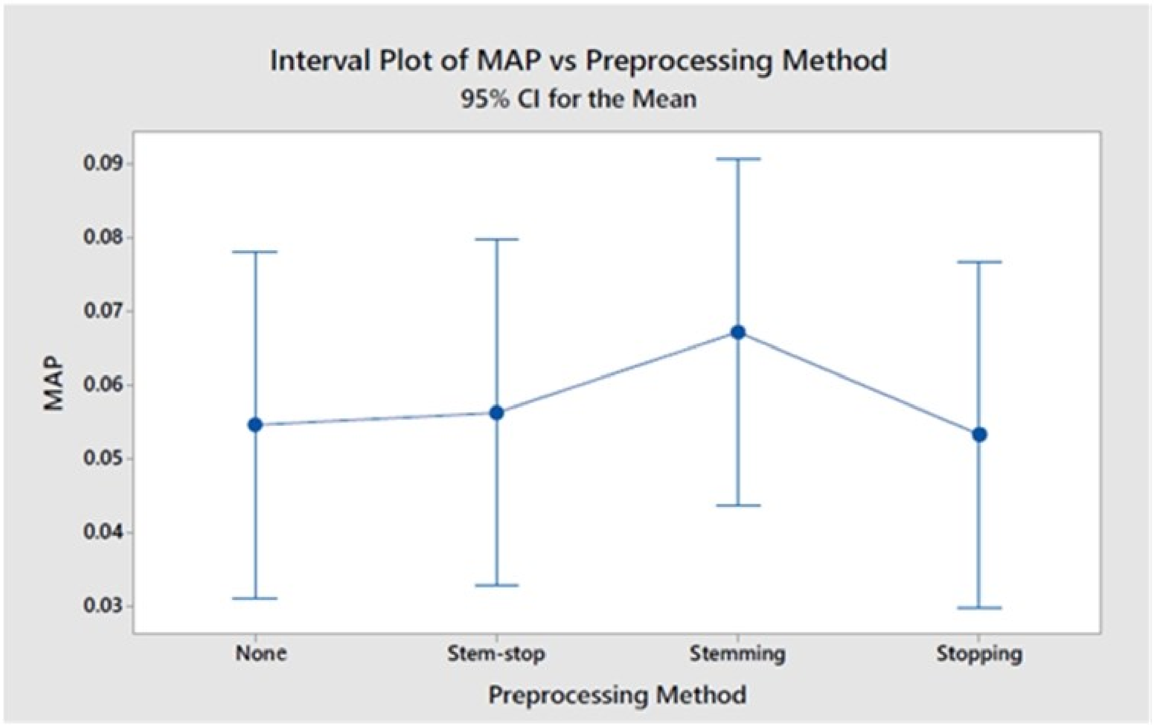

**Figure 5-18 MAP Statistical Test Results for LDA (Preprocessing Method)**

## 5.6 Case Study Challenges

We faced multiple challenges while conducting this empirical case study. We share here some of the challenges, hoping this can support future researchers and practitioners who plan to conduct similar empirical studies.

The main challenges can be summarized as follows:

- Data collection challenge: the major challenge was convincing an industrial partner to provide us with the data that was required for the study, including both the LL repository and the project artifacts data. The main problem was related to the confidentiality of our partner's data, which made it difficult to obtain their approval. To overcome this challenge, we had to communicate with a decision maker or senior management staff to convince them of the importance and the value of the study, so that they would release the data required for the study. Of course, this cost us time and effort.

- Evaluation process: to evaluate the LL classifiers that were provided, we had to compare the retrieved LL records to a reference map or gold set. This gold set should define the expected relevant LL records for each of the study queries, i.e., issue or risk records for the case study. It was challenging to ask our industrial partner PMs to be involved in constructing this gold set from scratch. Fortunately, the main researcher, the PhD candidate in this case, had extensive project management experience, so he took over the responsibility for constructing the gold set. Then, we asked one of the industrial partner PMs to simply review and validate the gold set, which was more achievable.

## 5.7 Scalability of the Automatic LL Recall Solution

Although the automatic LL classifiers have been constructed and the performance has been examined based on the dataset considered, our solution can be extended and applied to other organizations and datasets. In order to achieve that, practitioners can follow our process that is wrapped up as an inspiring framework as follows:

1- Construct the gold set based on the new dataset considered. As a hint, a method similar to that used in our study can be employed, where the judgements of multiple practitioners have been considered, then the baselined gold set has been based on the discussion and consensus of all the involved practitioners (please refer to Section 5.1.2).

2- Define the IR models considered. As a hint, as per our conclusion, the VSM and LSI models are more suitable than the LDA model for the LL recall context. Accordingly, practitioners can save the effort of examining the LDA model and directly proceed with constructing the IR classifier using VSM and LSI.

3- Define the experiment parameters. This includes defining the IR model parameter configurations, data preprocessing steps and performance metrics used.

4- Consider the new dataset. Practitioners should construct the IR classifiers, following the same method used in our experiment. This includes considering all the combinations of the IR models, preprocessing steps, and parameter configurations. As a hint, practitioners can follow our observations and conclusions of the best performance configurations recorded for our study.

Finally, benchmark the constructed LL classifiers and compare them according to the gold set, record the performance metrics, and choose the best achiever classifier to consider and deploy within the organization and dataset context.

## 5.8 Summary

Improving the awareness of a software organization's LL records can reform the decision making and project management processes. Providing an automatic process to support PMs in obtaining relevant LL records can improve the PMs' awareness of the organization's historical experiences. This is crucial for leveraging any potential opportunities and for mitigating any previous mistakes. We proposed a new automatic LL recall solution in Chapter 4. In this solution, we employed IR techniques for the first time within the software LL retrieval context.

In this chapter, we evaluated the effectiveness of the proposed solution, and sought answers to the research questions by conducting an empirical case study on a real dataset of industrial software projects. In the case study, we considered three state-of-the-art IR techniques, VSM, LSI and LDA, as well as the existing project artifacts, including the project issue and risk records. In addition, we verified statistically, using the Tukey's statistical test, the impact of considering different LL classifier parameter configurations on the classifiers' performance results. The impact of applying different preprocessing steps on the data records before constructing the LL classifiers was studied as well.

The case study results confirmed the effectiveness of the proposed solution and its ability to provide PMs with relevant LL in an automatic way and, thus, to eliminate the burden of the time and effort required to manually get the LL. The summary of our main findings is as follows:

- The best top-20 and MAP performance results were recorded for the VSM and LSI classifiers, while the LDA classifiers came next.

- Regarding the top-20, the best VSM classifier was configured using *tf-idf* for the *term weight*, *cosine* for the *similarity*, and *stemming* for the *preprocessing steps* of the LL and the queries. For the best LSI classifier, the configuration was the same for both *term weight* and *similarity* parameters, there were no preprocessing steps for the data records, and the *number of topics* was set to "128."

- Regarding the MAP performance results, the best classifiers for both VSM and LSI were configured using *sublinear tf-idf* for the *term weight*, *cosine* for the *similarity*, *stemming and stopping* for the *preprocessing steps*, as well as setting the *number of topics* to "128" for the LSI classifier.

- The statistical analysis of the different classifier configurations indicated the high impact of the configurations on the classifier performance. This was elicited from the significant difference between the performance of the best configured classifiers and the worst classifiers. As an example, for the VSM classifiers, the relative improvement between the best and worst classifiers was about 50% for the top-20 and more than 100% for MAP.

Moreover at the end of this chapter, we shared the threats to validity and some of the major challenges from our industrial case study regarding the data collection and the industrial partner's involvement. By sharing this, we aim to support practitioners in the industrial and software engineering community who would like to conduct similar studies in the future.

# Chapter 6

# 6 Can Hybridization Improve the Accuracy of Lessons Learned Recall: An Empirical Study Extension

In the previous empirical study in Chapter 5, we evaluated our automatic lessons learned (LL) recall solution using a real dataset from industry. The results of the case study proved the effectiveness of the solution by achieving an accuracy rate of about 70% in the case of top-k. In that study, we relied on some of the most popular information retrieval (IR) models from the literature to construct the LL classifiers. In addition, since our focus was limited to project management LL records, we relied on two of the existing and most influential project management artifacts, namely issues and risks, to actively invoke the constructed classifiers. Since these artifacts are already associated with the software development project lifecycle, there is no need for the manual involvement of project managers.

In this chapter, we will present an extension of the case study in Chapter 5.[4] In this extension, we considered one more research question RQ4, in addition to the three main research questions in Chapter 4 (See Section 4.2):

*RQ4: Can hybridization improve the LL recall accuracy?*

In order to answer this question, we constructed hybrid LL classifiers by combining multiple LL classifiers from the previous case study. The main motive for conducting such an extension was that although several domains studied the hybridization of classifiers [42][64], it was not studied in the LL recall context. In the extension study, we

[4] The work in this chapter is published in the First International Workshop on Professional Search (ProfS18), Co-located with ACM SIGIR 2018.

**Tamer Mohamed Abdellatif**, Luiz Fernando Capretz, and Danny Ho, "Searching for Relevant Lessons Learned Using Hybrid Information Retrieval Classifiers: A Case Study in Software Engineering," in *Joint Proceedings of the First International Workshop on Professional Search (ProfS2018); the Second Workshop on Knowledge Graphs and Semantics for Text Retrieval, Analysis, and Understanding (KG4IR); and the International Workshop on Data Search (DATA:SEARCH'18), Co-located with (ACM SIGIR 2018)*, 2018, Ann Arbor, Michigan, USA, pp. 12–17.

employed two popular hybridization methods and studied their impact on the performance of different classifier combinations.

## 6.1 Classifiers Hybridization

Different classifiers can perform in different ways in relation to the same dataset and inputs. This means that different classifiers can exhibit different errors and advantages. Thus, combining multiple classifiers together can lead either optimistically to better performance as they complement each other to avoid individual errors, or negatively to worse performance by distracting each other. This depends heavily on the chosen classifiers. Based on this information, we aim, in our case study, to combine multiple classifiers from the previous work to construct a hybrid classifier, and then study the impact of this combination on performance. We will compare the performance of the hybrid classifier to the performance of each combined classifier. The case study goal is to examine if we can achieve better performance by combining more than one classifier versus depending on each of the classifiers separately.

The hybridization calculation process is illustrated in Figure 6-1. As shown in the figure, the individual scores of the retrieved list—including the corresponding score for each LL record within the list—for each of the combined classifiers are forwarded to a hybrid technique. Based on the hybrid technique, the new hybrid score for each record, i.e., LL record in our case, is calculated. Then, the newly retrieved hybrid list is constructed by reordering the retrieved records according to the new hybrid scores. This retrieved hybrid list can be thought of as a retrieval list from a classifier, different than the original combined classifiers, which is constructed by combining the results of each combined classifier.

In order to evaluate the impact of hybridization on performance, the performance of the constructed hybrid classifier is compared to the performance of the highest performer classifier within the combined classifiers, as shown in Figure 6-1. The comparison is conducted by calculating the value of the relative performance improvement (RI) which will be described in detail in Section 6.4.

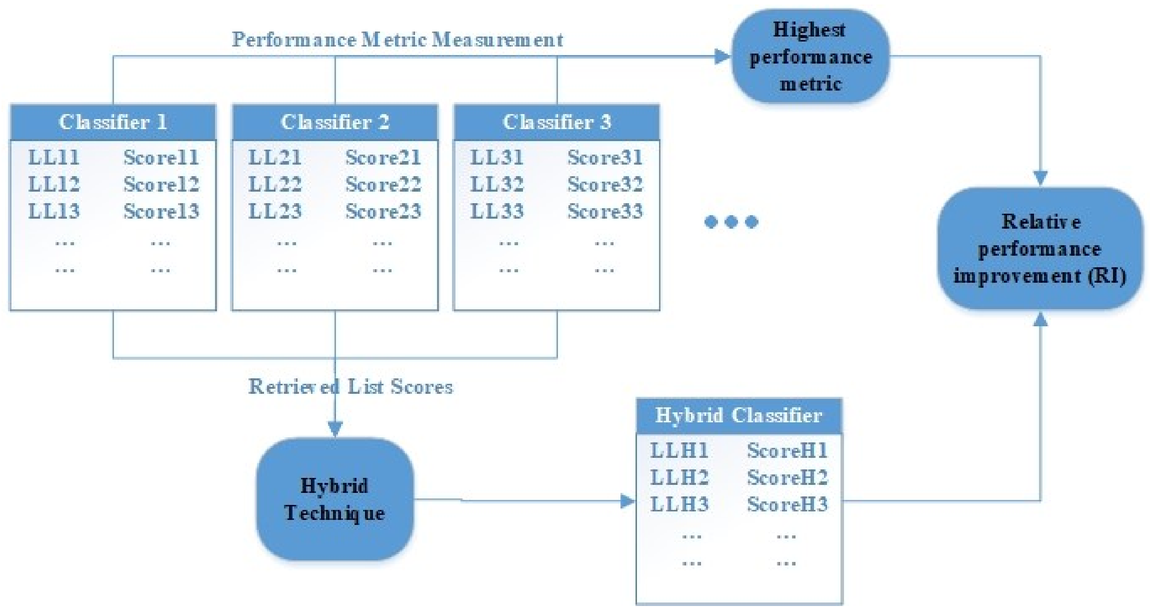

**Figure 6-1 Hybridization Calculation Process**

*Obs:*
*LLij: is the jth retrieved lessons learned record by the ith combined classifier, where j is the rank of this record within the retrieved list based on the given score.*
*Score ij: is the given score for the jth retrieved record by the ith combined classifier.*
*The hybrid classifier is considered as the classifier number H*

From the hybridization calculation process, it is clear, as described, that there are two main factors that affect the performance of hybridization: the employed hybrid technique and the selection of the combined classifiers. For this reason, in the following subsections, we will clarify both the hybrid techniques that we used to combine the classifiers and the selection criteria that we used to choose the classifiers that we combined.

## 6.2 Hybridization Techniques

In this study, we employed two popular hybridization methods from the software literature [42] namely Borda and Score Addition. We will describe the calculation of both methods in the following subsections.

### 6.2.1 Borda

The Borda technique is a rank-based technique. This means that it relies on the rank, (i.e., the order in the retrieved list of the retrieved item, the relevant LL in our case), within the classification results list from each individual classifier, to assign this item a rank-based score. So, for each retrieved LL item, the final rank or order within the hybrid retrieval list is the summation of the item ranks from each individual classifier retrieval list. Each of the item ranks is adjusted to the total number of items with non-zero rank score within each classifier retrieval list. This results in assigning the items with the highest Borda score an early appearance or low order in the final hybrid retrieval list. The Borda count can be calculated as stated in [42] as:

$$Borda\,(d_k) = \sum_{C_i \in C} M_i - r(d_k \mid C_i) + 1\,, \qquad [42]$$

where $d_k$ is the retrieved list item for which the Borda count is calculated, $C$ is the collection of the hybrid classifiers, $C_i$ is the ith classifier within the $C$ collection, $M_i$ is the number of retrieved items that received a non-zero score in the list retrieved by the classifier $C_i$, and $r(d_k \mid C_i)$ is the $d_k$ rank or order within the $C_i$ retrieved list [42].

### 6.2.2 Score Addition

The score addition technique relies on the item's weight, i.e., the score given by the individual classifiers. The total hybrid score of each retrieved item is calculated as the summation of the individual score of this item from each of the combined classifiers [42]. In order to avoid any mistaken bias to a certain classifier due to the weighting scale, the items' weights in each of the combined classifiers list are scaled to be within the same range of [0-1]. Accordingly, the individual item's score addition can be calculated as follows:

$$ScoreAddition(d_k) = \sum_{C_i \epsilon C} s(d_k \mid C_i) \qquad [42]$$

where $s(d_k \mid C_i)$ is the score of $d_k$ given by the classifier $C_i$ [42]. Finally, the items are placed in a descending order, based on their total score.

## 6.3 Hybrid Classifiers Selection

The selection of the combined classifiers has a crucial impact on the performance of the constructed hybrid classifier. For this reason, we tried to choose the classifiers that could positively complement each other. Thus, we chose the classifiers that had been exposed to different formats of the input data, because such classifiers would have a higher chance of coming up with different insights and conclusions regarding the dataset at hand, which we thought could improve their combined performance. That said, we decided to proceed with the classifiers that were constructed by applying the different input preprocessing step combinations.

As clarified in Chapter 4 (Section 4.3.1), the previous case study considered four different classifier subspaces or groups according to the preprocessing step combinations applied to the input data before it was forwarded to the IR model and before constructing the LL classifier. These preprocessing step combinations included applying none of the preprocessing steps, applying the stemming step, applying the stopping step, and applying both the stemming and stopping steps together. So, for each IR model, we considered a top performer classifier from each of the four classifier subspaces. This resulted in the selection of four classifiers from each of the VSM, LSI, and LDA models. The four selected classifiers included the top classifier when none of the preprocessing steps were applied, the top classifier when the stemming step was applied, the top classifier when the stopping step was applied, and finally the top performer classifier when both the stemming and stopping steps were applied together. In this experiment, we examined the performance of the hybrid classifiers constructed by combining the four selected classifiers of each IR model in pairs. In addition to studying these pairs of classifier combinations, we studied the performance of the combination of the four selected classifiers in each IR model as well. Finally, we combined all of the selected classifiers from all IR models together (four classifiers from each of the three IR models considered). All the classifier combinations are shown in Table 6-1.

## 6.4 Results

Since the goal of this case study extension is to investigate the impact of constructing a hybrid LL classifier by combining multiple LL classifiers from the previous study on performance, we constructed the hybrid classifiers following the selection criteria described in Section 6.3. The performance results for each of the constructed hybrid classifiers were recorded. Then, the relative performance improvement (RI) percentage was calculated. This was done by comparing the result of the hybrid classifier to that of the classifier with the highest performance among the individual classifiers within the combination set. The RI calculation is formulated as follows:

$$RI = \frac{P(HC) - HighestP\ (Combined\ Classifiers)}{HighestP\ (Combined\ Classifiers)}$$

where $P(HC)$ is the value of the performance metric $P$ for the hybrid classifier $HC$, and $HighestP()$ method returns the highest performance metric value among the performance values of the combined classifiers [42].

Since we considered two performance metrics in this study, top-20 and MAP, the results regarding each of these two metrics will be illustrated separately in the following sections.

## Table 6-1 Hybrid Classifiers

| | Top-20 Hybrid Classifiers | | MAP Hybrid Classifiers |
|---|---|---|---|
| LDA Top Classifiers | LDA_T1:LDA+32+None<br>LDA_T2:LDA+32+Stopping<br>LDA_T3:LDA+32+Stemming<br>LDA_T4:LDA+32+Stemming and stopping | LDA Top Classifiers | LDA_M1: LDA+32+None<br>LDA_M2: LDA+32+Stopping<br>LDA_M3: LDA+32+Stemming<br>LDA_M4: LDA+32+Stemming and stopping |
| **ID** | **Combined Classifiers** | **ID** | **Combined Classifiers** |
| CT1 | LDA_T1, LDA_T2 | CM1 | LDA_M1, LDA_M2 |
| CT2 | LDA_T2, LDA_T3 | CM2 | LDA_M2, LDA_M3 |
| CT3 | LDA_T1, LDA_T4 | CM3 | LDA_M1, LDA_M4 |
| CT4 | LDA_T2, LDA_T4 | CM4 | LDA_M2, LDA_M4 |
| CT5 | LDA_T3, LDA_T4 | CM5 | LDA_M3, LDA_M4 |
| CT6 | LDA_T1, LDA_T3 | CM6 | LDA_M1, LDA_M3 |
| CT7 | LDA_T1, LDA_T2, LDA_T3, LDA_T4 | CM7 | LDA_M1, LDA_M2, LDA_M3, LDA_M4 |
| **LSI Top Classifiers** | LSI_T1: LSI+TF-IDF+Cosine+128+None<br>LSI_T2: LSI+Sublinear+Cosine+64+Stopping<br>LSI_T3: LSI+Sublinear+Cosine+256+Stemming<br>LSI_T4: LSI+TF-IDF+Cosine+128+Stemming and stopping | **LSI Top Classifiers** | LSI_M1: LSI+TF-IDF+Cosine+256+None<br>LSI_M2: LSI+TF-IDF+Cosine+64+Sopping<br>LSI_M3: LSI+Sublinear+Cosine+64+Stemming<br>LSI_M4: LSI+Sublinear+Cosine+128+Stemming and stopping |
| **ID** | **Combined Classifiers** | **ID** | **Combined Classifiers** |
| CT8 | LSI_T2, LSI_T3 | CM8 | LSI_M1, LSI_M3 |
| CT9 | LSI_T3, LSI_T4 | CM9 | LSI_M3, LSI_M4 |
| CT10 | LSI_T1, LSI_T2 | CM10 | LSI_M1, LSI_M4 |
| CT11 | LSI_T1, LSI_T3 | CM11 | LSI_M2, LSI_M4 |
| CT12 | LSI_T1, LSI_T4 | CM12 | LSI_M2, LSI_M3 |
| CT13 | LSI_T2, LSI_T4 | CM13 | LSI_M1, LSI_M2 |
| CT14 | LSI_T1, LSI_T2, LSI_T3, LSI_T4 | CM14 | LSI_M1, LSI_M2, LSI_M3, LSI_M4 |
| **VSM Top Classifiers** | VSM_T1:VSM+Sublinear+Cosine+None<br>VSM_T2: VSM+Sublinear+Cosine+Stopping<br>VSM_T3: VSM+TF-IDF+Cosine+Stemming<br>VSM_T4: VSM+Sublinear+Cosine+Stemming and stopping | **VSM Top Classifiers** | VSM_M1: VSM+TF-IDF+Cosine+None<br>VSM_M2: VSM+TF-IDF+Cosine+Stopping<br>VSM_M3: VSM+TF-IDF+Cosine+Stemming<br>VSM_M4: VSM+Sublinear+Cosine+Stemming and stopping |
| **ID** | **Combined Classifiers** | **ID** | **Combined Classifiers** |
| CT15 | VSM_T1, VSM_T2 | CM15 | VSM_M1, VSM_M4 |
| CT16 | VSM_T1, VSM_T4 | CM16 | VSM_M2, VSM_M4 |
| CT17 | VSM_T2, VSM_T4 | CM17 | VSM_M3, VSM_M4 |
| CT18 | VSM_T1, VSM_T3 | CM18 | VSM_M1, VSM_M2 |
| CT19 | VSM_T2, VSM_T3 | CM19 | VSM_M1, VSM_M3 |
| CT20 | VSM_T3, VSM_T4 | CM20 | VSM_M2, VSM_M3 |
| CT21 | VSM_T1, VSM_T2, VSM_T3, VSM_T4 | CM21 | VSM_M1, VSM_M2, VSM_M3, VSM_M4 |
| CT22 | CT7, CT14, CT21 | CM22 | CM7, CM14, CM21 |

### 6.4.1 Top-K Results

The results for the hybrid classifiers that we considered and the impact on the top-20 are shown in Table 6-2. In the case of using the score addition method, the hybrid classifier results show either an improvement or no effect against the individual classifiers in about 77% of the cases considered. In other words, the score addition combination has led to a decrease in the performance in only five cases. Regarding the Borda method, there is an improvement or no effect in about 59% of the cases. The maximum improvement is 15% for the score addition method and 24% for the Borda method.

An important additional observation is that the combination performance has exceeded the 70% top-20, which was the top performance recorded among all the individual classifiers in the previous experimental work. For score addition, this is recorded in four cases where top-20 performance accuracies of 74% and 72% are recorded. In the case of Borda, this has been achieved in three cases where a top-20 of 72% is recorded. Also, it is important to highlight that the combination of the selected classifiers of all the IR models considered has led to an RI where the score addition results outperform or are comparable to the Borda results in most of the cases, at approximately 73%.

**Table 6-2 Top-20 Hybrid Classifiers Results**

| Combination ID | Top Individual Performance (%) | Score Addition | RI (%) | Borda Count | RI (%) |
|---|---|---|---|---|---|
| CT1 | 46 | 50 | 8 | 56 | 20 |
| CT2 | 46 | 52 | 12 | 57 | 24 |
| CT3 | 52 | 50 | -4 | 50 | -4 |
| CT4 | 52 | 54 | 4 | 56 | 7 |
| CT5 | 52 | 46 | -11 | 44 | -14 |
| CT6 | 41 | 46 | 14 | 44 | 9 |
| CT7 | 52 | 48 | -7 | 48 | -7 |
| CT8 | 69 | 69 | 0 | 70 | 3 |
| CT9 | 69 | 70 | 3 | 72 | 5 |
| CT10 | 70 | 67 | -5 | 70 | 0 |
| CT11 | 70 | 72 | 3 | 69 | -3 |
| CT12 | 70 | 74 | 5 | 69 | -3 |
| CT13 | 69 | 69 | 0 | 69 | 0 |
| CT14 | 70 | 70 | 0 | 70 | 0 |
| CT15 | 61 | 61 | 0 | 59 | -3 |
| CT16 | 61 | 70 | 15 | 65 | 6 |
| CT17 | 61 | 61 | 0 | 59 | -3 |
| CT18 | 70 | 65 | -8 | 63 | -11 |
| CT19 | 70 | 70 | 0 | 70 | 0 |
| CT20 | 70 | 72 | 3 | 72 | 3 |
| CT21 | 70 | 70 | 0 | 65 | -8 |
| CT22 | 70 | 72 | 3 | 72 | 3 |

Although the hybridization has not proven to be an improvement in all cases within this experiment, the number of the improved cases, especially the 77% of cases for score addition, is considered satisfactory and encourages the consideration of hybrid classifiers within the scope of LL retrieval context.

### 6.4.2 MAP Results

Table 6-3 shows the RI results in the case of the MAP performance metric. The results demonstrate either an improvement or no effect in the RI for about 81% of the cases using score addition. On the other hand, the improvement is not satisfactory in the case of the Borda method since the RI is negative for about 60% of the cases.

**Table 6-3 MAP Hybrid Classifiers Results**

| Combination ID | Top Individual Performance (%) | Score Addition | RI (%) | Borda Count | RI (%) |
|---|---|---|---|---|---|
| CM1 | 0.082 | 0.095 | 16 | 0.085 | 4 |
| CM2 | 0.096 | 0.096 | 0 | 0.106 | 10 |
| CM3 | 0.089 | 0.094 | 5 | 0.098 | 10 |
| CM4 | 0.089 | 0.090 | 1 | 0.084 | -6 |
| CM5 | 0.096 | 0.103 | 7 | 0.089 | -8 |
| CM6 | 0.096 | 0.102 | 6 | 0.114 | 18 |
| CM7 | 0.096 | 0.114 | 18 | 0.100 | 4 |
| CM8 | 0.175 | 0.182 | 4 | 0.172 | -2 |
| CM9 | 0.198 | 0.207 | 4 | 0.197 | -1 |
| CM10 | 0.198 | 0.200 | 1 | 0.186 | -6 |
| CM11 | 0.198 | 0.198 | 0 | 0.190 | -4 |
| CM12 | 0.194 | 0.199 | 3 | 0.196 | 1 |
| CM13 | 0.194 | 0.189 | -3 | 0.199 | 3 |
| CM14 | 0.198 | 0.199 | 0 | 0.199 | 0 |
| CM15 | 0.189 | 0.193 | 2 | 0.164 | -13 |
| CM16 | 0.189 | 0.169 | -11 | 0.156 | -18 |
| CM17 | 0.189 | 0.194 | 3 | 0.186 | -2 |
| CM18 | 0.131 | 0.149 | 14 | 0.130 | -1 |
| CM19 | 0.156 | 0.151 | -4 | 0.142 | -9 |
| CM20 | 0.156 | 0.175 | 12 | 0.156 | 0 |
| CM21 | 0.189 | 0.190 | 0 | 0.163 | -14 |
| CM22 | 0.198 | 0.195 | -2 | 0.160 | -19 |

Similar to the top-20 results, the same insight regarding the number of cases where the score addition outperformed or was comparable to the Borda, applies for the MAP results in about 81% of the cases.

## 6.5 Summary

In this chapter, we provided an extension of our previous empirical study regarding the construction of an automatic software management LL recall system. In this extension, we sought an answer for a research question, in addition to the questions answered in Chapter 5, that examined the impact of the hybridization of LL classifiers on performance. We relied on the existing LL classifiers from the previous study in Chapter 5 in constructing the hybrid classifiers. In the extension, we employed two combination techniques from literature in constructing the hybrid classifiers. A comparison was conducted between the performance of each hybrid classifier and the performance of the top performer from the combined individual classifiers.

Both top-K and MAP performance metrics were employed to measure the retrieval accuracy of the classifiers that were considered. The study results showed a relative improvement, or no effect, of the hybrid classifiers' performance against the individual classifiers' performance in about 77% of the cases of top-20 using the score addition method. On the other hand, the results regarding the MAP metric showed an improvement in about 81% of the cases when using score addition. Although, the improvement was not satisfactory in some cases, such as the MAP results in the case of using the Borda method, the overall results were encouraging and provided positive insights regarding employing IR classifiers hybridization within the LL recall context.

# Chapter 7

# 7 Summary and Future Work

In this thesis, we presented an innovative solution for improving the recall of software lessons learned (LL). To the best of our knowledge, this is the first time that a solution has employed information retrieval (IR) models within the LL recall context. Furthermore, we have proven the validity of the solution through an empirical case study using a real industrial dataset and performance metrics from the IR literature. In addition, we clarified how we automated the LL recall by constructing the search query on-the-fly using two of the existing project artifacts, issues and risks. We explained how our solution addresses the limitations of other studies, available from the literature, and eliminates the complication of manually searching LL repositories.

In Chapter 1, we described the context of the thesis and asserted the importance of the exploitation of an organization's knowledge. We clarified how the LL repository can be considered as one of the most highly valuable sources of knowledge and applicable analogs for an organization. In addition, we demonstrated the main motivation for the study and formulated the research questions. The motivation has two axes. First, we focused the solution, or research work, on supporting stakeholders other than software developers. Second, we sought to improve the exploitation of the organization's knowledge. We also clarified how we had defined our motivation. This was based on the insights from our comprehensive systematic literature review. Furthermore, we clearly stated the problem, i.e., the lack of automatic LL recall and how this can lead to overlooking existing LL records. That said, the main goal was to close this research gap by providing an automatic solution for LL recall based on IR techniques. We translated the problem statement to a research goal and formulated it into four research questions. Also, we listed the main research contributions and their mapping to the research questions in Section 1.3.

In Chapter 2, we explained in detail the protocol and methodology that we employed to conduct the systematic literature review (SLR) of software analytics (SA). This included the definition of the review questions, the search strategy, study selection, and data extraction and analysis (See Section 2.2.1). In the search strategy, we explained how we constructed the search query. We clarified the steps that we followed to improve this query until we came up with the search terms. Also, we listed the electronic libraries considered for the SLR. Regarding the studies selection, we defined the filtration criteria, for both inclusion and exclusion, and the quality assessment. The search resulted in 135 unique studies which were filtered and narrowed down to a final list of 19 primary studies. We extracted the needed data from these primary studies in order to come up with answers to the review questions. The results of the SLR provided informative insights and a vision of the SA state-of-the-art. We determined multiple research gaps, especially regarding the analyzed artifacts. Most of the primary studies analyzed only one artifact, which was source code in most cases. Furthermore, we defined some future research opportunities such as focusing on serving different stakeholders rather than only developers, as occurs in the majority of the existing studies. This can be beneficial to practitioners when deciding on their future projects and research problems. This was the first contribution of this thesis as we defined in the contribution list (See Section 1.3).

In Chapter 3, we clarified the main concepts and terms which were used in this thesis. This included the definition of LL and the review of the current research state of the LL recall. Also, we clarified the main concept of IR and provided some details regarding the three employed models, namely Vector Space Model (VSM), Latent Semantic Indexing (LSI), and Latent Dirichlet Allocation (LDA). The basic concepts regarding the configuration parameters for each of these models and the data preprocessing steps were provided.

We presented the research methodology in Chapter 4. We started by stating the problem of LL overlooking and the scarcity of the available LL recall solutions. We addressed this problem by articulating it in the research questions and goals. We clarified how we defined the research methodology and designed the case study to get answers to the research questions and validate the LL recall solution. In the research methodology, we explained how we designed and constructed the solution based on the IR techniques. Also, we demonstrated how the construction of the search queries was automated by dynamically building the queries using two of the existing project artifacts. The evaluation process, based on a real collected dataset from an industrial partner, was also defined. This included the definition of the two performance metrics considered, namely top-K and MAP. Also, we clarified the statistical test which we used to study the impact of different classifier configurations on performance.

We illustrated the execution and results of the case study in Chapter 5. We provided a description of the dataset used and how we constructed a gold set to use in the benchmarking of the classifiers performance. Also, we described the different classifier configurations and the applied data preprocessing steps combinations. The case study results showed a significant top-20 accuracy of 70% in the cases of VSM and LSI and a satisfactory MAP accuracy with the same models. In addition, an overall observation was that both VSM and LSI outperformed the LDA model. The LDA results were dissatisfactory and far away from the results of the other two models (See Sections 5.3 and 5.4). These results positively answered the first research question *RQ1* by proving the efficiency of employing IR models to automatically recall relevant LL (refer to Section 1.2 for the list of the research questions). Also, the results proved the efficiency of using both project management issues and risk register to dynamically construct the search queries, which bypassed the need for manually searching the LL repository and answered the second research question *RQ2*. The answer to the third research question *RQ3* was provided by the results of the statistical test which showed a high impact of the classifier configurations on performance. In addition, at the end of this chapter, we shared some of the case study threats to validity and challenges. We clarified how we dealt with each of these threats and how we overcame the challenges.

In Chapter 6, we extended the study by examining the ability of hybridization to improve the accuracy of classifiers. In order to achieve this, we sought an answer to the fourth research question *RQ4* by employing two of the hybridization techniques, namely Borda count and Score Addition. We constructed hybrid classifiers by combining individual classifiers, from the primary study in Chapter 5, using these two hybridization techniques. Also, we clarified the selection criteria of the combined classifiers. In choosing the selection criteria, our goal was to consider individual classifiers from different subspaces in order to boost each other's classification (See Section 6.3). Then, we compared the performance of the hybrid classifiers to that of the individual classifiers using relative performance improvement (RI). The results were significant, especially in the case of using the score addition technique where there was a performance improvement or no effect in about 77% of the cases for top-20 and 81% of the cases for MAP. Also, a relative improvement—up to 24%— was recorded for the top-20 using the Borda technique. Although, the results were not satisfactory in some cases, such as the MAP results when using the Borda method, the overall hybridization results provide positive insights and encouragement to employ hybridization in future IR studies within the LL recall context. By answering the four research questions, we provided the core contributions of this thesis as clarified in Section 1.3.

Since we conducted the first empirical study that considers applying IR techniques to tackle the automatic recall of software LL records for PMs, the results represent a value added to the state-of–the-art, and they can guide interested researchers, practitioners and organizations through the context of automatic LL retrieval.

## 7.1 Future Work

Since this work is the first, to the best of our knowledge, to apply IR techniques within the context of software LL retrieval, there are several promising avenues to extend the research as follows:

1. Considering other state-of-the-art IR ranking functions and models, such as Pivoted Length Normalization VSM [65], BM25F [66][67], and BM25+ [68]. This will extend our insights and boost the empirical evidence on the feasibility of employing those state-of-the-art functions within the software engineering domains, specifically the LL recall context.
2. Examining other weighing and similarity techniques, from the software literature. Regarding weighting techniques, they can include assigning different weights for different Part of Speech (PoS) tags as in [47]. There is no strong evidence from the literature that a specific part of speech can be more important than other parts in software engineering problems [47], rather it depends on the problem at hand. Therefore, it is important to examine the impact of considering PoS tags, such as

nouns and adjectives, on the performance of classifiers in the context of the problem at hand, i.e., LL recall in our case. Regarding similarity methods, other methods from the IR literature, such Manhattan distance can be examined as conducted by other IR studies [69].

3. Analyzing natural language patterns in the LL and project artifacts to determine if the patterns can be used to improve the retrieval accuracy.
4. Optimizing the selection of appropriate IR model configurations, based on the dataset and problem at hand, can be examined. Currently, this is an open research topic, especially regarding the optimization of the LDA model configurations. Recent software engineering research has revealed that text extraction from software engineering artifacts, such as source code, is more repetitive compared to the text extraction from regular natural language documents [70]. Therefore, it is important to examine the optimization of the IR model configurations based on the dataset at hand, especially for the LDA model since it gave poor results using the ad hoc parameter values recommended by the IR and natural language literature as shown in the case study. Optimization techniques, such as genetic algorithms, can be used to optimize the LDA parameters by maximizing an optimization function based on a similarity score between the inferred clusters, i.e., topics, as in [71].
5. Contacting more software organizations to collect more datasets. The new collected datasets can support an extension of our study for cross-organizational datasets. This has two main benefits. First, we can examine the validity of our study's overall observations and insights within different circumstances and organizational dataset contexts. Second, sampling techniques can be applied to construct a mixed dataset from cross-organizational datasets, and the feasibility of constructing a cross-organizational LL recall classifier based on this mixed data can be further studied.
6. Constructing a content-based recommender to serve long-term information needs of users, especially that the efficiency of recommendation systems has been highly examined for other software engineering problems and domains such as requirements elicitations [72] and adapted recommenders based on context awareness [73]. We designed our LL recall solution as an IR search engine based on the assumption that the need for relevant LL records, for the project at hand, is an ad-hoc information

need. However, our solution can be transformed to a content-based recommender to indicate the most relevant LL records based on the content similarity between the LL records and a text profile, instead of the queries in our study, of the project at hand [56].

7. Finally, conducting a utility study of the system usage, although it is challengeable, to evaluate the adoption of practitioners for our LL recall solution. Techniques from the IR domain, such as user interviewing [74] and studying the relationship between user clicks and the satisfaction level [75], can be employed. In addition, involving user feedback can support in transforming our solution into a collaborative-based recommender system based on the similarity analysis between different users' feedback [56].

# References

[1] G. A. Klein, *Sources of Power : How People Make Decisions*. MIT Press, 1999.

[2] J. W. Mullins and R. Komisar, *Getting to Plan B : Breaking Through to a Better Business Model*. Harvard Business Press, 2009.

[3] S. L. Pfleeger, "What Software Engineering Can Learn from Soccer," *IEEE Softw.*, vol. 19, no. 6, pp. 64–65, Nov. 2002.

[4] Project Management Institute, *A Guide to the Project Management Body of Knowledge (PMBOK Guide)*, 6th ed. Project Management Institute, 2017.

[5] M. B. Chrissis, M. Konrad, S. Shrum, and M. B. Chrissis, *CMMI for Development : Guidelines for Process Integration and Product Improvement*. Addison-Wesley, 2011.

[6] NASA, "*NASA Public Lessons Learned System*," 2018. [Online]. Available: https://llis.nasa.gov/. [Accessed: 28-Jun-2018].

[7] D. Zhang, Y. Dang, J. Lou, S. Han, H. Zhang, and T. Xie, "Software Analytics as a Learning Case in Practice," in *Proceedings of the International Workshop on Machine Learning Technologies in Software Engineering ( MALETS '11)*, 2011, pp. 55–58.

[8] A. E. Hassan, A. Hindle, P. Runeson, M. Shepperd, P. Devanbu, and S. Kim, "Roundtable: What's Next in Software Analytics," *IEEE Softw.*, vol. 30, no. 4, pp. 53–56, Jul. 2013.

[9] T. Menzies and T. Zimmermann, "Software Analytics: So What?," *IEEE Softw.*, vol. 30, no. 4, pp. 31–37, Jul. 2013.

[10] T. Menzies and T. Zimmermann, "The Many Faces of Software Analytics," *IEEE Softw.*, vol. 30, no. 5, pp. 28–29, Sep. 2013.

[11] T. Menzies and T. Zimmermann, "Goldfish Bowl Panel: Software Development Analytics," in *Proceedings of the 34th International Conference on Software Engineering (ICSE'12)*, 2012, pp. 1032–1033.

[12] B. Kitchenham and S. Charters, "Guidelines for Performing Systematic Literature Reviews in Software Engineering," *Engineering*, vol. 2, p. 1051, 2007.

[13] M. V. Brand, S. Roubtsov, and A. Serebrenik, "SQuAVisiT: A Flexible Tool for Visual Software Analytics," in *Proceedings of the 13th European Conference on Software Maintenance and Reengineering*, 2009, pp. 331–332.

[14] A. Gonzalez-Torres, R. Theron, F. J. Garcia-Penalvo, M. Wermelinger, and Y. Yu, "Maleku: An Evolutionary Visual Software Analysis Tool for Providing Insights into Software Evolution," in *Proceedings of the 27th IEEE International Conference on Software Maintenance (ICSM'11)*, 2011, pp. 594–597.

[15] E. Stroulia, I. Matichuk, F. Rocha, and K. Bauer, "Interactive Exploration of Collaborative Software-Development Data," in *Proceedings of the EEE International Conference on Software Maintenance*, 2013, pp. 504–507.

[16] D. Reniers, L. Voinea, O. Ersoy, and A. Telea, "The Solid* Toolset for Software Visual Analytics of Program Structure and Metrics Comprehension: From Research Prototype to Product," *Sci. Comput. Program.*, vol. 79, pp. 224–240, Jan. 2014.

[17] R. Minelli and M. Lanza, "Software Analytics for Mobile Applications--Insights & Lessons Learned," in *Proceedings of the 17th European Conference on Software Maintenance and Reengineering*, 2013, pp. 144–153.

[18] J. Lou, Q. Lin, R. Ding, Q. Fu, D. Zhang, and T. Xie, "Software Analytics for Incident Management of Online Services: An Experience Report," in *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering (ASE'13)*, 2013, pp. 475–485.

[19] C. Klammer and J. Pichler, "Towards Tool Support for Analyzing Legacy Systems in Technical Domains," in *Proceedings of the Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*, 2014, pp. 371–374.

[20] T. Taipale, M. Qvist, and B. Turhan, "Constructing Defect Predictors and Communicating the Outcomes to Practitioners," in *Proceedings of the ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, 2013, pp. 357–362.

[21] O. Baysal, R. Holmes, and M. W. Godfrey, "Developer Dashboards: The Need for Qualitative Analytics," *IEEE Softw.*, vol. 30, no. 4, pp. 46–52, Jul. 2013.

[22] P. M. Johnson, "Searching under the Streetlight for Useful Software Analytics," *IEEE Softw.*, vol. 30, no. 4, pp. 57–63, Jul. 2013.

[23] J. Czerwonka, N. Nagappan, W. Schulte, and B. Murphy, "CODEMINE: Building a Software Development Data Analytics Platform at Microsoft," *IEEE Softw.*, vol. 30, no. 4, pp. 64–71, Jul. 2013.

[24] J. Gong and H. Zhang, "BugMap: A Topographic Map of Bugs," in *Proceedings of the 9th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2013*, 2013, pp. 647–650.

[25] A. Miranskyy, B. Caglayan, A. Bener, and E. Cialini, "Effect of Temporal Collaboration Network, Maintenance Activity, and Experience on Defect Exposure," in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM '14*, 2014, pp. 1–8.

[26] R. Wu, H. Zhang, S.-C. Cheung, and S. Kim, "CrashLocator: Locating Crashing Faults Based on Crash Stacks," in *Proceedings of the International Symposium on Software Testing and Analysis - ISSTA 2014*, 2014, pp. 204–214.

[27] S. Han, Y. Dang, S. Ge, D. Zhang, and T. Xie, "Performance Debugging in the Large via Mining Millions of Stack Traces," in *Proceedings of the 34th International Conference on Software Engineering - ICSE 2012*, 2012, pp. 145–155.

[28] Y. Dubinsky, Y. Feldman, and M. Goldstein, "Where is the Business Logic?," in

*Proceedings of the 9th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2013*, 2013, pp. 667–670.

[29] N. Chen, J. Lin, S. C. H. Hoi, X. Xiao, and B. Zhang, “AR-miner: Mining Informative Reviews for Developers from Mobile App Marketplace,” in *Proceedings of the 36th International Conference on Software Engineering - ICSE 2014*, 2014, pp. 767–778.

[30] M. Mittal and A. Sureka, “Process Mining Software Repositories from Student Projects in an Undergraduate Software Engineering Course,” in *Companion Proceedings of the 36th International Conference on Software Engineering - ICSE Companion 2014*, 2014, pp. 344–353.

[31] G. Robles, J. M. González-Barahona, C. Cervigón, A. Capiluppi, and D. Izquierdo-Cortázar, “Estimating Development Effort in Free/Open Source Software Projects by Mining Software Repositories: A Case Study of OpenStack,” in *Proceedings of the 11th Working Conference on Mining Software Repositories - MSR 2014*, 2014, pp. 222–231.

[32] M. Choetkiertikul, H. K. Dam, T. Tran, and A. Ghose, “Predicting Delays in Software Projects Using Networked Classification (T),” in *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering - ASE’15*, 2015, pp. 353–364.

[33] B. Snyder and B. Curtis, “Using Analytics to Guide Improvement During an Agile–DevOps Transformation,” *IEEE Softw.*, vol. 35, no. 1, pp. 78–83, Jan. 2018.

[34] R. Weber, D. W. Aha, and I. Becerra-Fernandez, “Intelligent Lessons Learned Systems,” *Expert Syst. Appl.*, vol. 20, no. 1, pp. 17–34, Jan. 2001.

[35] R. O. Weber and D. W. Aha, “Intelligent Delivery of Military Lessons Learned,” *Decis. Support Syst.*, vol. 34, no. 3, pp. 287–304, Feb. 2003.

[36] W. Harrison, “A Software Engineering Lessons Learned Repository,” in *Proceedings of the 27th Annual NASA Goddard/IEEE Software Engineering Workshop*, 2002, pp. 139–143.

[37] C. Sary and W. Mackey, “A Case-based Reasoning Approach for the Access and Reuse of Lessons Learned,” in *Proceedings of the Fifth Annual International Symposium of the National Council on Systems Engineering*, 1995, vol. 1, pp. 249–256.

[38] R. Weber, D. W. Aha, K. Branting, J. R. Lucas, and I. Becerra-Fernandez, “Active Case-Based Reasoning for Lessons Delivery System,” in *Proceedings of the Florida Artificial Intelligence Research Society Conference - FLAIRS 2000*, 2000, pp. 170–174.

[39] M. M. Richter and R. O. Weber, *Case-Based Reasoning*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

[40] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge: Cambridge University Press, 2008.

[41] S. W. Thomas, A. E. Hassan, and D. Blostein, “Mining Unstructure Software

Repositories," in *Evolving Software Systems*, T. Mens, A. Serebrenik, and A. Cleve, Eds. Berlin, Heidelberg: Springer, 2014, pp. 139–162.

[42] S. W. Thomas, M. Nagappan, D. Blostein, and A. E. Hassan, "The Impact of Classifier Configuration and Classifier Combination on Bug Localization," *IEEE Trans. Softw. Eng.*, vol. 39, no. 10, pp. 1427–1443, 2013.

[43] A. T. Nguyen, T. T. Nguyen, J. Al-Kofahi, H. V. Nguyen, and T. N. Nguyen, "A Topic-based Approach for Narrowing the Search Space of Buggy Files from a Bug Report," in *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering - ASE 2011*, 2011, pp. 263–272.

[44] S. Rao and A. Kak, "Retrieval from Software Libraries for Bug Localization: A Comparative Study of Generic and Composite Text Models," in *Proceedings of the 8th Working Conference on Mining Software Repositories - MSR 2011*, 2011, pp. 43–52.

[45] M. Petrenko and V. Rajlich, "Concept Location Using Program Dependencies and Information Retrieval (DepIR)," *Inf. Softw. Technol.*, vol. 55, no. 4, pp. 651–659, Apr. 2013.

[46] T. Chen, S. W. Thomas, and A. E. Hassan, "A Survey on the Use of Topic Models When Mining Software Repositories," *Empir. Softw. Eng.*, vol. 21, no. 5, pp. 1843–1919, Oct. 2016.

[47] D. Falessi, G. Cantone, and G. Canfora, "Empirical Principles and an Industrial Case Study in Retrieving Equivalent Requirements via Natural Language Processing Techniques," *IEEE Trans. Softw. Eng.*, vol. 39, no. 1, pp. 18–44, Jan. 2013.

[48] R. A. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[49] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by Latent Semantic Analysis," *J. Am. Soc. Inf. Sci.*, vol. 41, no. 6, pp. 391–407, 1990.

[50] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet Allocation," *J. Mach. Learn. Res.*, vol. 3, no. Jan, pp. 993–1022, 2003.

[51] S. W. Thomas, "*LSCP: A Lightweight Source Code Preprocessor*," 2012. [Online]. Available: https://github.com/doofuslarge/lscp. [Accessed: 28-Jun-2018].

[52] S. W. Thomas, "Mining Software Repositories with Topic Models," Tech. Report, *Sch. Comput. Queen's Univ.*, 2012.

[53] A. K. McCallum, "*Mallet: A Machine Learning for Language Toolkit*," 2002. [Online]. Available: http://mallet.cs.umass.edu. [Accessed: 28-Jun-2018].

[54] S. W. Thomas, "*Lucene-lda: Use Latent Dirichlet Allocation (LDA) in Apache Lucene*," 2012. [Online]. Available: https://github.com/stepthom/lucene-lda. [Accessed: 28-Jun-2018].

[55] X. Wei and W. B. Croft, "LDA-based Document Models for Ad-hoc Retrieval," in

*Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2006, pp. 178–185.

[56] C. Zhai and S. Massung, *Text Data Management and Analysis: A Practical Introduction to Information Retrieval and Text Mining*. New York, NY, USA: Association for Computing Machinery and Morgan & Claypool, 2016.

[57] J. W. Tukey, "Comparing Individual Means in the Analysis of Variance," *Biometrics*, vol. 5, no. 2, pp. 99–114, 1949.

[58] J. W. Tukey, "The Philosophy of Multiple Comparisons," *Stat. Sci.*, vol. 6, no. 1, pp. 100–116, 1991.

[59] J. W. Tukey, D. R. Brillinger, H. Braun, L. V Jones, and D. R. Cox, *The Collected Works of John W. Tukey: Multiple Comparions*. Taylor & Francis, 1984.

[60] M. F. Porter, "An Algorithm for Suffix Stripping," *Program*, vol. 14, no. 3, pp. 130–137, Mar. 1980.

[61] Apache, "Apache Lucene: Java-based Indexing and Search Technology," *Apache Lucene: Java-based indexing and search technology*, 2004. [Online]. Available: https://lucene.apache.org/. [Accessed: 30-Jun-2018].

[62] R. Rehurek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, 2010, pp. 45–50.

[63] B. Cleary, C. Exton, J. Buckley, and M. English, "An Empirical Analysis of Information Retrieval Based Concept Location Techniques in Software Comprehension," *Empir. Softw. Eng.*, vol. 14, no. 1, pp. 93–130, Feb. 2009.

[64] E. Kocaguneli, T. Menzies, and J. Keung, "On the Value of Ensemble Effort Estimation," *IEEE Trans. Softw. Eng.*, vol. 38, no. 6, pp. 1403–1416, Nov. 2012.

[65] A. Singhal, C. Buckley, and M. Mitra, "Pivoted Document Length Normalization," *SIGIR Forum*, vol. 51, no. 2, pp. 176–184, Aug. 2017.

[66] S. Robertson, H. Zaragoza, and M. Taylor, "Simple BM25 Extension to Multiple Weighted Fields," in *Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management*, 2004, pp. 42–49.

[67] S. Robertson and H. Zaragoza, "The Probabilistic Relevance Framework: BM25 and Beyond," *Found. Trends Inf. Retr.*, vol. 3, no. 4, pp. 333–389, Apr. 2009.

[68] Y. Lv and C. Zhai, "Lower-bounding Term Frequency Normalization," in *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, 2011, pp. 7–16.

[69] J. Herrera, B. Poblete, and D. Parra, "Learning to Leverage Microblog Information for QA Retrieval," in *Proceedings of the European Conference on Information Retrieval*, 2018, pp. 507–520.

[70] A. Hindle, E. T. Barr, Z. Su, M. Gabel, and P. Devanbu, "On the Naturalness of Software," in *Proceedings of the 34th International Conference on Software Engineering - ICSE'12*, 2012, pp. 837–847.

[71] A. Panichella, B. Dit, R. Oliveto, M. Di Penta, D. Poshynanyk, and A. De Lucia, "How to Effectively Use Topic Models for Software Engineering Tasks? An Approach Based on Genetic Algorithms," in *Proceedings of the 35th International Conference on Software Engineering - ICSE'13*, 2013, pp. 522–531.

[72] C. Palomares, X. Franch, and D. Fucci, "Personal Recommendations in Requirements Engineering: The OpenReq Approach," in *IProceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality*, 2018, pp. 297–304.

[73] D. Bachmann, K. Grolinger, H. ElYamany, W. Higashino, M. Capretz, M. Fekri, and B. Gopalakrishnan, "(CF)2 Architecture: Contextual Collaborative Filtering," *Inf. Retr. J.*, May 2018.

[74] J. Garcia-Gathright, B. St. Thomas, C. Hosey, Z. Nazari, and F. Diaz, "Understanding and Evaluating User Satisfaction with Music Discovery," in *Proceedings of the 41st International ACM SIGIR Conference on Research & Development in Information Retrieval - SIGIR '18*, 2018, pp. 55–64.

[75] H. Lu, M. Zhang, and S. Ma, "Between Clicks and Satisfaction: Study on Multi-Phase User Preferences and Satisfaction for Online News Reading," in *Proceedings of the 41st International ACM SIGIR Conference on Research & Development in Information Retrieval - SIGIR '18*, 2018, pp. 435–444.

# Appendices

## Appendix A: Examples of Lessons Learned, Issues and Risks

Note: sensitive information was removed from all of the following examples.

### A) Lessons Learned Examples

| **Example 1** | |
|---|---|
| Problem | Discussion in program review meetings was sometimes getting too emotional and long-winded. |
| Recommendations | Participants should understand their roles in the meeting, better control by chairperson is also suggested. |
| **Example 2** | |
| Problem | Module X of System Y was not developed or published in advance to the teams until the last minute, without sufficient instruction. |
| Recommendations | Instructions on Module X should be standardized and disseminated throughout the organization early enough. |
| **Example 3** | |
| Opportunity | Completed version and training on Module Z with excellent feedback from customers. |
| Recommendations | This can be used as a model for future versions:<br>- good planning sessions with customers prior to release, regular dialogue to update status of preparation for the release, exchange of test plan/cases/etc., readiness for the version from both sides. |
| **Example 4** | |
| Problem | Too much context switching amongst team members |
| Recommendations | Since this is unavoidable due to attrition, separation, career planning, etc., constant update to organization chart within tools team is needed. Team members are to share domain knowledge, back each other up as part of organization planning. |

## B) Issues Examples

| |
|---|
| Delay in signing requirement and design documents by client. |
| There is no availability of a technical writer. |
| Additional ramp up effort and constant re-clarification of roles and responsibilities due to context switching. |
| Project Contract is not clear and has not been signed yet |

## C) Risks Examples

| |
|---|
| Source code is at client side with no remote access. If no appropriate backup and labeling are processed, then an issue can happen or code loss can occur. |
| If there is delay in requirement document sign off by customer as planned on <date>, this can lead to delay of schedule and can affect milestone dates and resources travel dates. |
| If roles of different stakeholders are not set clear, then this will impact the scoping and requirements sign off. |
| If there is any issue in issuing an entry Visa for the team leader, then this can lead to delay of schedule and can affect milestone dates and resources travel dates |

## D) Mapping Relevant Lessons Learned to a Query Example

| **Query** | **Relevant Lessons Learned** |
|---|---|
| Additional ramp up effort and constant re-clarification of roles and responsibilities due to context switching. | Problem: Too much context switching amongst team members.<br>Recommendations:<br>Since this is unavoidable due to attrition, separation, career planning, etc., constant update to organization chart within tools team is needed. Team members are to share domain knowledge, back each other up as part of organization planning. |

## Appendix B: Top-20 88 Classifiers Results

### VSM:

| Term weight | Similarity method | Preprocessing steps | Top-20 (%) |
|---|---|---|---|
| tf-idf | Cosine | Stemming | 70 |
| Sublinear tf-idf | Cosine | Stemming | 69 |
| Sublinear tf-idf | Cosine | None | 61 |
| Sublinear tf-idf | Cosine | Stemming and stopping | 61 |
| tf-idf | Cosine | None | 61 |
| tf-idf | Cosine | Stemming and stopping | 61 |
| Boolean | Cosine | Stemming and stopping | 57 |
| Boolean | Overlap | Stemming and stopping | 57 |
| Sublinear tf-idf | Overlap | Stemming and stopping | 57 |
| tf-idf | Overlap | Stemming and stopping | 57 |
| Sublinear tf-idf | Cosine | Stopping | 54 |
| Sublinear tf-idf | Overlap | Stemming | 54 |
| tf-idf | Cosine | Stopping | 54 |
| tf-idf | Overlap | Stopping | 54 |
| Boolean | Cosine | Stopping | 52 |
| Boolean | Overlap | Stopping | 52 |
| Boolean | Cosine | Stemming | 52 |
| Sublinear tf-idf | Overlap | None | 52 |
| Sublinear tf-idf | Overlap | Stopping | 52 |
| tf-idf | Overlap | None | 52 |
| tf-idf | Overlap | Stemming | 52 |
| Boolean | Overlap | Stemming | 50 |
| Boolean | Cosine | None | 46 |
| Boolean | Overlap | None | 46 |

### LSI:

| Term weight | Similarity method | Preprocessing steps | Number of topics | Top-20 (%) |
|---|---|---|---|---|
| tf-idf | Cosine | None | 128 | 70 |
| Sublinear tf-idf | Cosine | None | 128 | 69 |
| Sublinear tf-idf | Cosine | Stemming | 256 | 69 |
| tf-idf | Cosine | None | 256 | 69 |
| tf-idf | Cosine | Stemming | 128 | 69 |
| tf-idf | Cosine | Stemming and stopping | 128 | 69 |

| | | | | |
|---|---|---|---|---|
| Sublinear tf-idf | Cosine | None | 256 | 67 |
| Sublinear tf-idf | Cosine | Stopping | 64 | 67 |
| Sublinear tf-idf | Cosine | Stemming | 128 | 67 |
| Sublinear tf-idf | Cosine | Stemming and stopping | 128 | 67 |
| tf-idf | Cosine | Stopping | 64 | 67 |
| tf-idf | Cosine | Stemming | 256 | 67 |
| Sublinear tf-idf | Cosine | Stopping | 128 | 65 |
| tf-idf | Cosine | Stopping | 128 | 65 |
| Boolean | Cosine | Stopping | 64 | 63 |
| Sublinear tf-idf | Cosine | Stopping | 256 | 63 |
| Sublinear tf-idf | Cosine | Stemming and stopping | 64 | 63 |
| tf-idf | Cosine | Stopping | 256 | 63 |
| tf-idf | Cosine | Stemming and stopping | 64 | 63 |
| Boolean | Cosine | Stemming and stopping | 64 | 61 |
| Sublinear tf-idf | Cosine | Stemming and stopping | 256 | 61 |
| tf-idf | Cosine | Stemming | 64 | 61 |
| tf-idf | Cosine | Stemming and stopping | 256 | 61 |
| Boolean | Cosine | Stemming and stopping | 128 | 59 |
| Sublinear tf-idf | Cosine | None | 64 | 59 |
| Sublinear tf-idf | Cosine | Stemming | 32 | 59 |
| Sublinear tf-idf | Cosine | Stemming | 64 | 59 |
| tf-idf | Cosine | None | 32 | 59 |
| tf-idf | Cosine | None | 64 | 59 |
| Sublinear tf-idf | Cosine | None | 32 | 57 |
| tf-idf | Cosine | Stemming | 32 | 57 |
| Boolean | Cosine | Stopping | 128 | 56 |
| Boolean | Cosine | Stopping | 256 | 56 |
| Boolean | Cosine | Stemming | 128 | 56 |

| | | | | |
|---|---|---|---|---|
| Sublinear tf-idf | Cosine | Stemming and stopping | 32 | 56 |
| tf-idf | Cosine | Stemming and stopping | 32 | 56 |
| Sublinear tf-idf | Cosine | Stopping | 32 | 54 |
| tf-idf | Cosine | Stopping | 32 | 54 |
| Boolean | Cosine | None | 256 | 52 |
| Boolean | Cosine | Stemming | 256 | 52 |
| Boolean | Cosine | Stemming and stopping | 32 | 52 |
| Boolean | Cosine | Stemming and stopping | 256 | 52 |
| Boolean | Cosine | Stopping | 32 | 50 |
| Boolean | Cosine | Stemming | 32 | 50 |
| Boolean | Cosine | Stemming | 64 | 50 |
| Boolean | Cosine | None | 128 | 48 |
| Boolean | Cosine | None | 64 | 44 |
| Boolean | Cosine | None | 32 | 43 |

**LDA:**

| Preprocessing steps | Number of topics | Top-20 (%) |
|---|---|---|
| Stemming and stopping | 32 | 52 |
| Stopping | 32 | 46 |
| Stemming and stopping | 64 | 46 |
| None | 32 | 41 |
| Stemming | 32 | 41 |
| Stemming | 64 | 39 |
| None | 64 | 35 |
| None | 128 | 35 |
| Stopping | 256 | 35 |
| Stopping | 64 | 28 |
| Stopping | 128 | 26 |
| Stemming | 128 | 26 |
| Stemming and stopping | 128 | 26 |
| Stemming | 256 | 22 |
| None | 256 | 19 |
| Stemming and stopping | 256 | 19 |

## Appendix C: MAP 88 Classifiers Results

**VSM:**

| Term weight | Similarity method | Preprocessing steps | MAP |
|---|---|---|---|
| Sublinear tf-idf | Cosine | Stemming and stopping | 0.189 |
| tf-idf | Cosine | Stemming and stopping | 0.188 |
| tf-idf | Cosine | Stemming | 0.156 |
| Sublinear tf-idf | Cosine | Stemming | 0.153 |
| tf-idf | Overlap | Stemming and stopping | 0.151 |
| Sublinear tf-idf | Overlap | Stemming and stopping | 0.150 |
| Boolean | Cosine | Stemming and stopping | 0.140 |
| Boolean | Overlap | Stemming and stopping | 0.140 |
| tf-idf | Cosine | None | 0.131 |
| Sublinear tf-idf | Cosine | None | 0.126 |
| tf-idf | Overlap | Stemming | 0.124 |
| Sublinear tf-idf | Overlap | Stemming | 0.122 |
| tf-idf | Cosine | Stopping | 0.121 |
| Sublinear tf-idf | Cosine | Stopping | 0.119 |
| Sublinear tf-idf | Overlap | Stopping | 0.118 |
| tf-idf | Overlap | Stopping | 0.117 |
| Boolean | Cosine | Stopping | 0.114 |
| Boolean | Overlap | Stopping | 0.114 |
| Boolean | Cosine | Stemming | 0.101 |
| Boolean | Overlap | Stemming | 0.101 |
| tf-idf | Overlap | None | 0.099 |
| Sublinear tf-idf | Overlap | None | 0.095 |
| Boolean | Cosine | None | 0.082 |
| Boolean | Overlap | None | 0.081 |

**LSI:**

| Term weight | Similarity method | Preprocessing steps | Number of topics | MAP |
|---|---|---|---|---|
| Sublinear tf-idf | Cosine | Stemming and stopping | 128 | 0.198 |
| tf-idf | Cosine | Stemming and stopping | 128 | 0.198 |
| tf-idf | Cosine | Stopping | 64 | 0.194 |
| Sublinear tf-idf | Cosine | Stopping | 64 | 0.194 |
| Sublinear tf-idf | Cosine | Stemming and stopping | 64 | 0.183 |
| Boolean | Cosine | Stopping | 64 | 0.181 |
| tf-idf | Cosine | Stemming and stopping | 256 | 0.180 |
| tf-idf | Cosine | Stemming and stopping | 64 | 0.179 |
| Sublinear tf-idf | Cosine | Stopping | 128 | 0.177 |
| Sublinear tf-idf | Cosine | Stemming and stopping | 256 | 0.176 |
| Sublinear tf-idf | Cosine | Stemming | 64 | 0.175 |
| Sublinear tf-idf | Cosine | Stopping | 256 | 0.174 |
| tf-idf | Cosine | Stopping | 256 | 0.174 |
| tf-idf | Cosine | Stemming | 64 | 0.171 |
| Sublinear tf-idf | Cosine | Stemming | 128 | 0.169 |
| tf-idf | Cosine | Stopping | 128 | 0.168 |
| Sublinear tf-idf | Cosine | Stemming | 256 | 0.167 |
| tf-idf | Cosine | Stemming | 256 | 0.167 |
| tf-idf | Cosine | Stemming and stopping | 32 | 0.167 |
| Sublinear tf-idf | Cosine | Stemming and stopping | 32 | 0.165 |
| Boolean | Cosine | Stopping | 128 | 0.164 |
| Boolean | Cosine | Stemming and stopping | 128 | 0.164 |
| tf-idf | Cosine | Stemming | 32 | 0.164 |
| Sublinear tf-idf | Cosine | Stemming | 32 | 0.163 |
| tf-idf | Cosine | None | 256 | 0.163 |
| tf-idf | Cosine | Stemming | 128 | 0.162 |
| Sublinear tf-idf | Cosine | None | 256 | 0.161 |
| Sublinear tf-idf | Cosine | None | 128 | 0.157 |
| tf-idf | Cosine | None | 128 | 0.157 |
| Boolean | Cosine | Stemming and stopping | 256 | 0.154 |
| Sublinear tf-idf | Cosine | None | 64 | 0.153 |
| tf-idf | Cosine | None | 64 | 0.153 |
| Boolean | Cosine | Stopping | 256 | 0.149 |
| Boolean | Cosine | Stemming and stopping | 64 | 0.144 |
| tf-idf | Cosine | Stopping | 32 | 0.135 |
| Sublinear tf-idf | Cosine | Stopping | 32 | 0.132 |
| Boolean | Cosine | Stemming and stopping | 32 | 0.130 |
| Boolean | Cosine | Stopping | 32 | 0.125 |

| | | | | |
|---|---|---|---|---|
| tf-idf | Cosine | None | 32 | 0.124 |
| Sublinear tf-idf | Cosine | None | 32 | 0.119 |
| Boolean | Cosine | Stemming | 128 | 0.116 |
| Boolean | Cosine | Stemming | 256 | 0.115 |
| Boolean | Cosine | None | 256 | 0.111 |
| Boolean | Cosine | Stemming | 64 | 0.111 |
| Boolean | Cosine | None | 128 | 0.107 |
| Boolean | Cosine | None | 64 | 0.096 |
| Boolean | Cosine | Stemming | 32 | 0.086 |
| Boolean | Cosine | None | 32 | 0.085 |

**LDA:**

| Preprocessing steps | Number of topics | MAP |
|---|---|---|
| Stemming | 32 | 0.096 |
| Stemming and stopping | 32 | 0.089 |
| None | 32 | 0.082 |
| Stopping | 32 | 0.075 |
| Stemming and stopping | 64 | 0.066 |
| Stemming | 128 | 0.065 |
| Stemming | 64 | 0.059 |
| Stopping | 256 | 0.058 |
| None | 128 | 0.057 |
| None | 64 | 0.049 |
| Stemming | 256 | 0.049 |
| Stopping | 128 | 0.044 |
| Stemming and stopping | 128 | 0.040 |
| Stopping | 64 | 0.036 |
| None | 256 | 0.031 |
| Stemming and stopping | 256 | 0.030 |

## Appendix D: Tukey's HSD Statistical Test Results

Obs: the overall insights from the results in the case of 95% confidence level hold for both 90% and 99% confident levels.

Note: for the following results, any difference than the results of 95% confidence level is highlighted in red.

### A) 90% Confidence Level

**Top-20 Results:**

| VSM | | | LSI | | | LDA | | |
|---|---|---|---|---|---|---|---|---|
| Group | Mean (%) | Preprocessing steps | Group | Mean (%) | Preprocessing steps | Group | Mean (%) | Preprocessing steps |
| A | 59 | Stemming and stopping | A | 60 | Stopping | A | 36 | Stemming and stopping |
| A | 58 | Stemming | A | 60 | Stemming and stopping | A | 34 | Stopping |
| A | 54 | None | A | 60 | Stemming | A | 32 | None |
| A | 53 | Stopping | A | 58 | None | A | 32 | Stemming |
| Group | Mean (%) | Similarity | Group | Mean (%) | Number of topics | Group | Mean (%) | Number of topics |
| A | 58 | Cosine | A | 63 | 128 | A | 45 | 32 |
| B | 53 | Overlap | A | 61 | 256 | AB | 37 | 64 |
| | | | AB | 60 | 64 | BC | 28 | 128 |
| | | | B | 54 | 32 | C | 24 | 256 |
| Group | Mean (%) | Term weight | Group | Mean (%) | Term weight | | | |
| A | 58 | tf-idf | A | 63 | tf-idf | | | |
| A | 57 | Sublinear tf-idf | A | 63 | Sublinear tf-idf | | | |
| A | 52 | Boolean | B | 53 | Boolean | | | |

**MAP Results:**

| VSM | | | LSI | | | LDA | | |
|---|---|---|---|---|---|---|---|---|
| Group | Mean | Preprocessing steps | Group | Mean | Preprocessing steps | Group | Mean | Preprocessing steps |
| A | 0.159 | Stemming and stopping | A | 0.170 | Stemming and stopping | A | 0.067 | Stemming |
| B | 0.126 | Stemming | A | 0.164 | Stopping | A | 0.056 | Stemming and stopping |
| B | 0.117 | Stopping | AB | 0.147 | Stemming | A | 0.055 | None |
| B | 0.102 | None | B | 0.132 | None | A | 0.053 | Stopping |
| Group | Mean | Similarity | Group | Mean | Number of topics | Group | Mean | Number of topics |
| A | 0.135 | Cosine | A | 0.161 | 128 | A | 0.085 | 32 |
| A | 0.117 | Overlap | A | 0.161 | 64 | B | 0.053 | 64 |
| | | | AB | 0.158 | 256 | B | 0.051 | 128 |
| | | | B | 0.133 | 32 | B | 0.042 | 256 |
| Group | Mean | Term weight | Group | Mean | Term weight | | | |
| A | 0.136 | tf-idf | A | 0.167 | Sublinear tf-idf | | | |
| A | 0.134 | Sublinear tf-idf | A | 0.166 | tf-idf | | | |
| A | 0.109 | Boolean | B | 0.127 | Boolean | | | |

## B) 99% Confidence Level

**Top-20 Results:**

| **VSM** | | | **LSI** | | | **LDA** | | |
|---|---|---|---|---|---|---|---|---|
| Group | Mean (%) | Preprocessing steps | Group | Mean (%) | Preprocessing steps | Group | Mean (%) | Preprocessing steps |
| A | 59 | Stemming and stopping | A | 60 | Stopping | A | 36 | Stemming and stopping |
| A | 58 | Stemming | A | 60 | Stemming and stopping | A | 34 | Stopping |
| A | 53 | None | A | 60 | Stemming | A | 32 | None |
| A | 53 | Stopping | A | 58 | None | A | 32 | Stemming |
| Group | Mean (%) | Similarity | Group | Mean (%) | Number of topics | Group | Mean (%) | Number of topics |
| A | 58 | Cosine | A | 63 | 128 | A | 45 | 32 |
| A | 53 | Overlap | AB | 61 | 256 | AB | 37 | 64 |
| | | | AB | 60 | 64 | AB | 28 | 128 |
| | | | B | 54 | 32 | B | 24 | 256 |
| Group | Mean (%) | Term weight | Group | Mean (%) | Term weight | | | |
| A | 58 | tf-idf | A | 63 | tf-idf | | | |
| A | 57 | Sublinear tf-idf | A | 63 | Sublinear tf-idf | | | |
| A | 52 | Boolean | B | 53 | Boolean | | | |

**MAP Results:**

| VSM | | | LSI | | | LDA | | |
|---|---|---|---|---|---|---|---|---|
| Group | Mean | Preprocessing steps | Group | Mean | Preprocessing steps | Group | Mean | Preprocessing steps |
| A | 0.159 | Stemming and stopping | A | 0.170 | Stemming and stopping | A | 0.067 | Stemming |
| AB | 0.126 | Stemming | AB | 0.164 | Stopping | A | 0.056 | Stemming and stopping |
| B | 0.117 | Stopping | AB | 0.147 | Stemming | A | 0.055 | None |
| B | 0.102 | None | B | 0.132 | None | A | 0.053 | Stopping |
| Group | Mean | Similarity | Group | Mean | Number of topics | Group | Mean | Number of topics |
| A | 0.135 | Cosine | A | 0.161 | 128 | A | 0.085 | 32 |
| A | 0.117 | Overlap | A | 0.161 | 64 | B | 0.053 | 64 |
| | | | A | 0.158 | 256 | B | 0.051 | 128 |
| | | | A | 0.133 | 32 | B | 0.042 | 256 |
| Group | Mean | Term weight | Group | Mean | Term weight | | | |
| A | 0.136 | tf-idf | A | 0.167 | Sublinear tf-idf | | | |
| A | 0.134 | Sublinear tf-idf | A | 0.166 | tf-idf | | | |
| A | 0.109 | Boolean | B | 0.127 | Boolean | | | |

# Curriculum Vitae

| | |
|---|---|
| **Name:** | Tamer Mohamed Abdellatif Mohamed |
| **Post-secondary Education and Degrees:** | Western University<br>London, Ontario, Canada<br>2014-2018 Ph.D. (GPA 95%) |
| | Ain Shams University<br>Cairo, Egypt<br>2005-2011 M.Sc. |
| | Ain Shams University<br>Cairo, Egypt<br>1999-2004 B.Eng. |
| **Honours and Awards:** | Ontario Graduate Scholarship (OGS)<br>2017-2018 |
| **Related Work Experience** | Teaching and Research Assistant<br>Western University<br>2014-2018 |
| | Project Manager<br>ITWorx, Hewlett-Packard (HP)<br>2010-2014 |
| | Technical Leader and Software Developer<br>Asset Technology Group<br>2004-2009 |

**Thesis Related Publications and Presentations:**

**Tamer Mohamed Abdellatif**, Luiz Fernando Capretz, and Danny Ho, "Searching for Relevant Lessons Learned Using Hybrid Information Retrieval Classifiers: A Case Study in Software Engineering," in *Joint Proceedings of the First International Workshop on Professional Search (ProfS2018); the Second Workshop on Knowledge Graphs and Semantics for Text Retrieval, Analysis, and Understanding (KG4IR); and the International Workshop on Data Search (DATA:SEARCH'18), Co-located with ACM Special Interest Group on Information Retrieval (ACM SIGIR 2018)*, 2018, Ann Arbor, Michigan, USA, pp. 12–17.

**Tamer Mohamed Abdellatif,** Luiz Fernando Capretz, and Danny Ho, “Active Recall of Software Lessons Learned for Software Project Managers,” *Inf. Softw. Technol. (IST)*. March 2018. (Under review)

**Tamer Mohamed Abdellatif,** Luiz Fernando Capretz, and Danny Ho, “Software Analytics to Software Practice: A Systematic Literature Review,” in *Proceedings of the 37th International Conference on Software Engineering (ICSE) Workshop on BIG Data Software Engineering (BIGSE)*, 2015, Florence, Italy, pp. 30–36.