

2018-01-03

# An Analogy Based Technique for Predicting the Vector Impact of Software Change Requests

Kabeer, Shaikh Jeeshan

---

Kabeer, S. J. (2018). An Analogy Based Technique for Predicting the Vector Impact of Software Change Requests (Master's thesis, University of Calgary, Calgary, Canada). Retrieved from <https://prism.ucalgary.ca>. <http://hdl.handle.net/1880/106289>

*Downloaded from PRISM Repository, University of Calgary*

UNIVERSITY OF CALGARY

An Analogy Based Technique for Predicting the Vector Impact of Software Change Requests

by

Shaikh Jeeshan Kabeer

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES  
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE  
DEGREE OF MASTER OF SCIENCE

GRADUATE PROGRAM IN COMPUTER SCIENCE

CALGARY, ALBERTA

January, 2018

© Shaikh Jeeshan Kabeer 2018

# Abstract

Managing the impact of software change request (CR) requires proper prediction to minimize cost and enhance quality. Traditional techniques only identify source code entities likely to be impacted by the implementation of CR. However, looking at code entity is not enough. To ensure maximum utility and return on investment, decision-makers also need to consider effort and duration requirements. It is more challenging for Minimum Viable Products which need to implement essential CR promptly. This thesis proposes an analogy based reasoning approach called Vector Change Impact Analysis which can generate file, effort and duration predictions simultaneously. Case-studies on industrial and open source projects highlight effectiveness of VCIA. The proposed approach is integrated into an analytics dashboard, for software company Brightsquid Inc. Analyzing performance results under different settings, VCIA achieves 67% file impact prediction recall. For effort and duration prediction, achieved accuracy is 44% and 30% respectively. Applying VCIA to complement expert knowledge has resulted in improved planning and organization, higher customer satisfaction and better forecasting of future returns.

## Acknowledgements

This thesis work would not have been possible without the help and guidance provided by so many. First and foremost I would like to extend my heartfelt gratitude towards my supervisor Dr. Guenther Ruhe. He has been a constant source of guidance and motivation. Dr. Ruhe's immense knowledge, expertise and industry experience in Software Engineering and management, has made the journey of this Master's thesis to be a genuinely gratifying learning experience.

I extend my sincere appreciation towards the examiners and neutral chair of my MSc defense committee, Dr. Raymond Patterson, Dr. Behrouz Far and Dr. Jorg Denzinger. Thank you for kindly agreeing to be on my examination board. Your valuable feedback and recommendations have helped immensely to enhance this thesis.

I would like to mention here The Alberta Innovates Technology Futures (AITF) graduate scholarship, which helped me a great deal towards supporting myself. It allowed me to worry less on monetary aspects and more on research.

I would like to take this opportunity to thank the instructors of all my courses at the University of Calgary. The tools, concepts, and techniques that I learned in these classes were indispensable towards my research and this thesis. I would also like to recognize the support and assistance rendered by all members of Software Engineering and Decision Support (SEDS) laboratory. In particular, I would like to acknowledge the guidance of Maleknaz, Rezaul and Didar, whose excellent research abilities are ideal roadmaps for all graduate students.

Last but not the least, this thesis would probably not have been possible without the unconditional love and support of my wife Tasnuba and family back home. We have lived worlds apart in the last two years, but your affection has remained with me always. My good friends Lazima, Imtiaz, Mamun, Masroor, Jubair, Nabi and Tafseer, you have honored me with your friendship. My sincere thanks to my family in Calgary - my aunt, brother, sister, nephew, and niece. All of you have helped turn this strange land into something resembling Home.

# Table of Contents

<b>Abstract</b>	ii
<b>Acknowledgements</b>	iii
Table of Contents	iv
List of Tables	vi
List of Figures	vii
List of Symbols	ix
1 Introduction	1
1.1 Motivation	1
1.2 Challenges	3
1.3 Research Objectives	5
1.4 Solution overview	6
1.5 Thesis structure	7
2 Background & Literature Review	9
2.1 Background	9
2.1.1 Iterative development	9
2.1.2 Software change	10
2.1.3 Vector Change Impact Analysis (VCIA)	12
2.1.4 Mining Software Repository (MSR)	13
2.2 Literature Review	14
2.2.1 Existing techniques: code impact, effort and duration prediction	14
2.2.2 Analogy based problem solving	22
2.2.3 State of industry evaluation	27
2.2.4 State of tool support	27
2.3 Summary	29
3 Proposed approach: Vector Change Impact Analysis (VCIA)	30
3.1 Vector Change Impact Analysis (VCIA)	30
3.1.1 Knowledge base representation	31
3.1.2 Search and retrieval	34
3.1.3 Selection	35
3.1.4 Prediction	35
3.2 Vector impact prediction techniques: algorithm and description	36
3.2.1 Bag of Words (BOW)	36
3.2.2 Bag of Words & Latent Dirichlet Allocation (BOW & LDA)	37
3.2.3 Bag of Words & File Coupling (BOW & CO)	41
3.2.4 Alignment with objectives	42
3.2.5 Evaluation measures	42
3.3 Summary	44
4 Case Studies	46
4.1 Case study 1: Evaluation of VCIA on real-world industry projects	46
4.1.1 Context of the study	47
4.1.2 Data collection	50
4.1.3 Initialization & parameter setting	50

4.1.4	Case study objective . . . . .	51
4.1.5	Value of predicting vector impact of change . . . . .	51
4.1.6	Analysis of results . . . . .	53
4.1.7	Improvement for Brightsquid . . . . .	60
4.2	Threats to validity . . . . .	62
4.2.1	Construct validity . . . . .	62
4.2.2	Conclusion validity . . . . .	63
4.2.3	Internal validity . . . . .	64
4.2.4	External validity . . . . .	64
4.3	Case study 2: Evaluating VCIA in OSS . . . . .	64
4.3.1	Case study objective . . . . .	64
4.3.2	Context of the study . . . . .	65
4.3.3	Data collection . . . . .	65
4.3.4	Analysis of results . . . . .	66
4.4	Threats to validity . . . . .	74
4.5	Comparison with Literature . . . . .	75
4.6	Summary . . . . .	80
5	VCIA tool support . . . . .	81
5.1	Introduction . . . . .	81
5.2	Overview of tool development . . . . .	82
5.3	Architecture . . . . .	83
5.4	Tool modules . . . . .	85
5.4.1	Ripple Effect Analysis . . . . .	86
5.4.2	Commit Traceability Analysis . . . . .	88
5.5	Summary . . . . .	89
6	Summary & Future research . . . . .	91
6.1	Summary of contributions . . . . .	91
6.1.1	VCIA approach for predicting files impacted, effort and duration of change request . . . . .	91
6.1.2	Application and validation with industrial case study . . . . .	91
6.1.3	Tool development for vector impact prediction . . . . .	92
6.2	Limitations & Applicability . . . . .	92
6.3	Future research scope . . . . .	93
A	Additional figures, tables from experimental results and copyright form . . . . .	95
	Bibliography . . . . .	102

## List of Tables

2.1	Existing literature related to code impact prediction. Each literature is described by: Technique (approach used for impact prediction, IR = Information Retrieval, CO = File co-change), Data source (data used for processing), Effort indicator(whether effort can be predicted), Duration indicator(whether duration can be predicted) and Industry case study (whether literature has been validated with industrial data). . . . .	15
2.2	Existing literature related to effort and duration prediction. Each literature is described by: Technique (approach used to make prediction, KNN = K Nearest Neighbour), Type(the types of CR supported), Code Impact Indicator(whether code impact can be predicted) and Industry case study (whether literature has been validated with industrial data). . . . .	23
2.3	Tools for code impact analysis, where each tool is evaluated against multiple functionality defined on the left. . . . .	29
3.1	Data attributes from CR and code repositories utilized by VCIA. For each attribute the <i>Name</i> , <i>Repository</i> , <i>Type</i> and <i>Availability</i> are shown. . . . .	33
4.1	List of case studies and corresponding Research Questions. . . . .	47
4.2	Time frame and number of change requests for vector impact prediction in industry projects [43]. . . . .	50
4.3	Accuracy of file impact prediction for the three proposed techniques applied to Mail and Dental projects and reported for different number of file recommendations $F$ [43]. . . . .	54
4.4	Accuracy of effort and duration prediction for BOW and BOW & LDA techniques applied to Mail and Dental projects and reported for varying number of similar change request recommendations $N$ . . . . .	57
4.5	Time frame and number of change requests for vector impact prediction in OSS projects. . . . .	66
4.6	Accuracy of file impact prediction for the three proposed techniques applied to Apache and DSpace projects and reported for varying number of file recommendations $F$ . . . . .	67
4.7	Accuracy of effort and duration prediction for BOW and BOW & LDA techniques applied to Apache and DSpace projects and reported for varying number of change request recommendations $N$ . . . . .	69
4.8	Performance comparison of file impact prediction techniques from literature and proposed approach, shown for different number of file recommendations $F$ . $P$ and $R$ stand for Precision and Recall measures respectively. “-” indicate unavailable evaluation data (adapted from [43]). . . . .	78
4.9	Comparison of effort and duration prediction results from literature and VCIA. “-” indicate unavailable evaluation data. . . . .	79
5.1	List of tools and technologies used for VCIA tool development . . . . .	82
A.1	Number of entity analyzed in VCIA and literature studies. . . . .	98

## List of Figures and Illustrations

2.1	Figure showing Iterative software development . . . . .	10
3.1	Process outline of this thesis (adapted from [43]). . . . .	45
4.1	Process model at Brightsquad. . . . .	48
4.2	Change management process at Brightsquad (adapted from [43]) . . . . .	49
4.3	Recall of file impact prediction by applying the three proposed techniques on industry projects Mail (M) and Dental (D), in dependence to the number of file recommendations $F$ [43]. . . . .	53
4.4	Precision of file impact prediction by applying the three proposed techniques on industry projects Mail (M) and Dental (D), in dependence to the number of file recommendations $F$ [43]. . . . .	55
4.5	Recall of file impact prediction for Bug VS Non-Bug CR, achieved with BOW & CO applied on Dental project, in dependence to the number of file recommendations $F$ [43]. . . . .	56
4.6	Recall of file impact prediction for Bug VS Non-Bug CR, achieved with BOW & CO applied on Mail project, in dependence to the number of file recommendations $F$ [43]. . . . .	57
4.7	Effort prediction accuracy applying BOW and BOW & LDA on Mail & Dental projects, in dependence to the number of change request recommendations $N$ . . . .	58
4.8	Duration prediction accuracy applying BOW and BOW & LDA on Mail & Dental projects, in dependence to the number of change request recommendations $N$ . . . .	59
4.9	Recall of file impact prediction by applying the three proposed techniques on Apache and DSpace, in dependence to the number of file recommendations $F$ . . . .	66
4.10	Precision of file impact prediction by applying the three proposed techniques on Apache and DSpace, in dependence to the number of file recommendations $F$ . . . .	68
4.11	Duration prediction accuracy applying BOW and BOW & LDA on Apache & DSpace projects, in dependence to the number of change request recommendation $N$ . . . . .	70
4.12	Recall of BOW & CO file impact prediction applied on OSS and industry projects, in dependence to the number of file recommendations $F$ . . . . .	71
4.13	Precision of BOW & CO file impact prediction applied on OSS and industry projects, in dependence to the number of file recommendations $F$ . . . . .	72
4.14	Accuracy of duration prediction by applying BOW on OSS and industry projects, in dependence to the number of change request recommendations $N$ . . . . .	73
4.15	Accuracy of effort prediction by applying BOW & LDA on OSS and industry projects, in dependence to the number of change request recommendations $N$ . . . .	74
4.16	Effort prediction accuracy applying BOW and BOW & LDA on Apache & DSpace projects, in dependence to the number of change request recommendations $N$ . . . .	75
5.1	Tool homepage . . . . .	81
5.2	Tool architecture layers. . . . .	85
5.3	Input of <i>Ripple Effect Analysis</i> module for vector impact recommendation. . . . .	86



5.4	Output of <i>Ripple Effect Analysis</i> showing <i>Impacted Files</i> recommendation (sensitive values hidden to preserve industry confidentiality). . . . .	87
5.5	Output of <i>Ripple Effect Analysis</i> showing predicted effort and duration value recommendation. . . . .	87
5.6	Output of <i>Ripple Effect Analysis</i> showing <i>Similar Issues</i> recommendation (sensitive values hidden to preserve industry confidentiality). . . . .	88
5.7	Input screen of <i>Commit Traceability Analysis</i> module. . . . .	89
5.8	Sample output of <i>Commit Traceability Analysis</i> (sensitive values hidden to preserve industry confidentiality). . . . .	90
A.1	F10 scores of file impact prediction by applying the three proposed techniques on industry projects Mail(M) and Dental(D), in dependence to the number of file recommendations $F$ [43]. . . . .	95
A.2	Ratio of change request types for Mail project. . . . .	96
A.3	Ratio of change request types for Dental project. . . . .	96
A.4	F10 scores of file impact prediction by applying the three proposed techniques on industry projects Apache and DSpace, in dependence to the number of file recommendations $F$ . . . . .	97
A.5	Recall of file impact prediction for different values of $CO$ threshold, by applying the BOW & CO on Mail, in dependence to the number of file recommendations $F$ . . . . .	97
A.6	Recall of file impact prediction for <i>Stemming</i> and <i>Lemmatization</i> operations, by applying the BOW & CO on Mail, in dependence to the number of file recommendations $F$ . . . . .	99
A.7	Accuracy of effort prediction for <i>Mean Median</i> value calculation, by applying BOW & LDA on Mail, in dependence to the number of change request recommendations $N$ . . . . .	99
A.8	Copyright form of paper [43] - Page 1 . . . . .	100
A.9	Copyright form of paper [43] - Page 2 . . . . .	101

## List of Symbols, Abbreviations and Nomenclature

Symbol	Definition
AWS	Amazon Web Services
ABR	Analogy Based Reasoning
BOW	Bag of Words
CIA	Change Impact Analysis
CR	Change Request
CO	Coupling
EC2	Elastic Compute Cloud
FLT	Feature Location Techniques
FPM	Frequent Pattern Mining
FPA	Function Point Analysis
ID	Identification
IR	Information Retrieval
IRT	Issue Resolution Time
KNN	K Nearest Neighbour
LDA	Latent Dirichlet allocation
LOC	Lines Of Code
LOOCV	Leave One Out Cross Validation
MRE	Mean Relative Error
MVP	Minimum Viable Product
MSR	Mining Software Repository
OS	Operating System
OSS	Open Source Software
RQ	Research Question

TDM	Term Document Matrix
TF-IDF	Term Frequency-Inverse Document Frequency
TM	Topic Modeling
VCIA	Vector Change Impact Analysis

# Chapter 1

## Introduction

### 1.1 Motivation

Addressing software requirements is of paramount importance to the success of software. Requirements represent intended goals and objectives that should be fulfilled by the software. They define the capabilities and functionality of the software system. Requirement generation is a continuous process, it is therefore not possible to satisfy all requirements in a single release of the software. Alterations and improvements are being requested all the time. These appear in the form of Change Requests (CRs), which are also alternatively referred to as *tickets* or *issues*. CRs appear during all parts of system development and maintenance [59]. They originate from changes in stakeholder demand, market, tools and technology [26, 66]. Change requests need to be handled efficiently to reduce software cost and improve quality [95]. The following lists the primary motivations of this thesis:

**Aid Change Request decision making:** Software development takes place in constrained environments with limitations on resources, mainly time and human expertise available per release. As the implementation of requests consumes these finite resources, decision-makers need to choose a CR subset. The selection of CRs for a particular release is an investment of these limited resources, and the primary objective is to maximum utility and return on investment. A crucial part of this decision making needs to take into account effort and duration required (for any CR). Thus, the approach taken by traditional impact analysis techniques to predict the impact on the code base is not enough. Of course, when planning any change file impact need to be considered. However, decision-makers will also need to evaluate the time that

will be required to implement the change. Another critical factor is the amount of effort that will be necessary to fulfill changes.

**Support Minimum Viable Product (MVP) development:** MVP development concentrate on providing a set of critical features which appear promising to potential users. The aim is to design, develop and release accelerated products to hit the market quickly. MVP help to elicit customer feedback and also secure position in the market [78]. The product is incrementally refined with multiple iterations, with more complete set of features being added over time, making the system complete and mature. Compared to large and medium organizations, small emerging companies, typically start-ups operate with additional resource constraints. They focus on satisfying Minimum Viable Products (MVP), which is an encouraging and popular way to enter the marketplace in relatively shorter time [29]. The fast-paced nature of development usually translates to more and frequent releases, as a result estimation decisions need to be made more often. In addition, constrained resources need to be allocated in a manner which enhance return and improve predictions. Based on this discussion, it is clear that change management is particularly challenging for MVP. Thus, the need for better comprehension and management of change request is significantly importance for MVP software development.

**Reduce cognitive load:** Modern software involve vast numbers of entities interacting in an elaborate setup. Due to the sheer volume of data associated with real-life software systems, it is impossible to identify and handle changes manually. Integrating computational intelligence with human expertise has become popular in software engineering [80]. In particular, researchers and practitioners have developed tools to address a whole spectrum of software engineering activities, starting with requirement management, software design, coding and testing to evaluations [79]. Various literature have highlighted improvement in software decision making with the in-

tegration of tools, and computational intelligence in general [19, 92]. The second chapter of this thesis, provide a detailed account on the state of tool support of change impact analysis. This analysis have uncovered limitations with availability and applicability of existing tools for change impact assessment. Thus, it would be beneficial if tool backed techniques can be developed which can address these drawbacks.

## 1.2 Challenges

Software development is a complex process which generates a complex web of interacting entities. Usually, products are offered as variants on different OS and platform variations. The result is a complicated system with different functionality implemented across large number of files. The scarcity of time and resource make the task of software change decision making particularly challenging. The following list highlights some of the key challenges:

- **Duration & effort consideration:** Existing literature emphasize on the identification of impacted code components [19]. Starting with an initiating entity such as CR, file or method the focus is the identification of file, method or class that might be impacted by the corresponding initiating entity. Also, some of the techniques utilize class data, which restricts the usage to object-oriented systems. As already stated before, looking at changes to code entities only does not aid proper decision making, because decision-makers cannot make an informed decision on the resource allocation and the perceived return on investment. Looking further into existing literature [57] revealed that there is no consideration of the effort or duration that might be required to implement the change. Thus decision making on software change becomes more challenging for developers and product managers. There is some literature which focuses on predicting the effort and duration based on CR textual content and additional attributes. But these approaches have not been

integrated or tested together with code impact analysis techniques. Moreover, most of the literature is directed towards duration prediction for *Bug* type change requests and very few on effort estimation from change request directly.

- **Lack of industrial validation:** The techniques established in existing literature have mostly been experimented using open source software (OSS) project data. OSS projects are different compared to industrial projects, the latter being more strict in terms of processes and policies. People working in OSS are usually volunteers and the release window are usually flexible. On the other hand employees working in the industry receive remunerations and have specific release timing window. These fundamental differences to OSS, lead to significant disparity in change requests being implemented and the corresponding modified files. As a result, data stored in repositories are also significantly different. Thus code impact analysis, as well as effort and duration prediction techniques, need to be validated using both industry and OSS data, to ensure that results are repeatable and transferable [71].
- **Lack of tool support:** There are some tools which perform code impact analysis. However, these tools are not generic for many reasons. Firstly, four of the five tools analyzed, perform code impact analysis in Java<sup>1</sup> programs only. Secondly, a majority of the tools act as plugins for Eclipse<sup>2</sup> IDE, which would force developers to use specific IDE for development and impact analysis. It may be counterproductive pushing developers towards particular IDE or platform. Moreover, some of the tools require human intervention, which makes impact analysis less efficient and more labor intensive. Finally, tools have been primarily developed with the developer in mind and possess little industry validation.

---

<sup>1</sup><https://www.java.com/>

<sup>2</sup><https://www.eclipse.org/ide/>

- **Reliance on high churn code data:** Existing literature generate knowledge base using two categories of data i.e., *CR* and *code* [72]. CR data refers to primarily textual attributes of change requests such as CR Title and CR Description. The second category, code data refer to various textual entities existing in the code, such as code comments, identifier names, method names, class names, etc. CR data is relatively stable because it does not go through many modifications, after the request is created in the repository. Conversely, code data can go through big rapid changes, as the code files continue to be regularly modified and updated, by developers working on the different functionality. Thus, code changes would need tracking and corresponding knowledge base would require major updates in short intervals.

### 1.3 Research Objectives

The discussion in the preceding two sections formulate the motivation for this thesis. Consequently, the main goal of this thesis can be defined as “ **To design and implement a technique which can predict the files impacted, effort and duration required to implement a change request**”. Based on the main goal, the following research objectives are defined:

- RO1: Review existing literature related to change impact analysis, effort and duration prediction.
  - RO1.1: Analyze and classify existing literature based on decision indicators, data source, issue type and techniques.
  - RO1.2: Evaluate the state of industry validation.
  - RO1.3: Evaluate the state of tool support.
- RO2: Design proposed techniques which can predict the impact with respect to a change request.



- RO2.1: Predict files impacted by change request.
  - RO2.2: Predict effort required to fulfill change request.
  - RO2.3: Predict duration required to fulfill change request.
- RO3: Evaluation of proposed techniques towards predicting file impact, effort and duration required for change request.
  - RO3.1: Case study on real-world industry data.
  - RO3.2: Case study on open-source data.
- RO4: Development of an integrated tool based on the proposed technique.
  - RO4.1: Development of a tool which can predict files impacted as well as effort and duration required.
  - RO4.2: Validated by using in an industrial setting and overcome limitations of existing impact analysis tools.

## 1.4 Solution overview

Towards fulfilling the objectives RO 2,3 and 4 stated in the previous section, this thesis proposes three textual similarity based techniques for file impact prediction, which follow *Analogy Based Reasoning* method of inferring from previous problems and solutions. Knowledge base, which store past history (problems and solution) is constructed using past CR textual data, and information on the files they impacted. For a query, the textual similarity is used to define analogy i.e., similarity with solved instances, to make predictions for the query. The three techniques all use Bag of Words (BOW) [62] - a multi-set representation of textual terms with repetitions allowed. The first technique use textual similarity of BOW representation of change requests to make the prediction. Textual similarity is widely used in literature [20,21] for making predictions. The second technique integrate textual similarity with topic similarity, using topics generated from Latent

Dirichlet allocation (LDA), to make the prediction. LDA is another technique being widely used in software engineering [23]. Finally, the third technique generate file coupling and integrates that with textual similarity. Coupling was utilized in a number of impact prediction literature [32, 45]. The first two techniques are used for file impact, effort and duration prediction, while the third model (textual similarity with coupling) is used only for impact prediction.

The techniques were evaluated using two case studies, performed under diverse settings. The first case study was performed at the software development unit of **Brightsquid**, located in Calgary, Canada. It is a start-up company with a MVP focus and are keen to improve efficiency through software analytics driven solutions. The effort was part of Natural Sciences and Engineering Research Council of Canada (NSERC) industry collaboration (CRD)<sup>3</sup> project. Subset of the results and findings from the first case study, appear in the paper [43]. The second case study was performed on two well known OSS projects [85]. In both cases, performance of proposed techniques are evaluated with respect to file impact, effort and duration prediction. The performance are analyzed under different perspectives, including comparison of results obtained with industry and OSS data, which helps to ensure consistency and reliability. This thesis also reports on a tool, based on the proposed techniques, that has been integrated into an analytics dashboard for Brightsquid.

## 1.5 Thesis structure

The following list outlines the structure of thesis work.

- *Chapter 1:* Statement of thesis motivation, outlines of the challenges, list of research objectives and finally the solution approach.
- *Chapter 2:* Description of the background, core concepts and literature review of existing approaches.

---

<sup>3</sup>NSERC-CRD-486636-2015

- *Chapter 3:* Introduction of the preliminary concepts, process, data and evaluation measures related of the proposed techniques.
- *Chapter 4:* Reporting results from two case studies conducted in real-world industry and OSS projects.
- *Chapter 5:* Outline and description of the tool developed following the techniques defined for file impact, effort and duration prediction.
- *Chapter 6:* Summary of the contributions, highlight of key results and outline of future research directions.

## **Chapter 2**

### **Background & Literature Review**

#### **2.1 Background**

This chapter begins with an overview of software development methodology and software change. Later existing literature related to file impact, effort and duration prediction with respect to change requests are presented. Existing literature is analyzed in terms of the extent to which file impact techniques consider effort and duration requirements, and vice-versa. Subsequently, the state of industrial validation and tool support availability is examined, towards providing change impact decision making. The chapter is intended to provide introduction to the fundamentals of the related concepts and provide justifications of proposed techniques outlined in the next chapter.

##### **2.1.1 Iterative development**

Software development methodology typically follows either linear or repetitive approach. In linear approach such as waterfall model [54], development activities proceed in a sequential pattern. Key activities such as analysis, design, implementation, testing and release happen sequentially. However, this is too restrictive and not feasible in a dynamically changing landscape such is software development. A more popular approach is the repetitive software development methodology called iterative software development [55]. Compared to linear models, development activities do not stop after one cycle, rather it keeps on repeating the activities in multiple cycles or iterations. Figure 2.1 outline the operating characteristics of iterative development. This is more popular because it allows change to take place more easily, as releasing software in multiple iterations gives the ability to continuously integrate stakeholder requirements. Also, planning and risk assessment become easier to integrate, as activities are spread across iterations, which can be adjusted more easily to changing requirements. However iterative approach presents an additional challenge -

requirement changes need to be handled on a continuous basis as development is always ongoing. Stakeholders will come up with new requirements due to various external and internal scenarios. The inherent expectation is that these changing requirements will be accommodated in upcoming releases. Subsequently, change impact decision making is also continuous. Decision makers will constantly need to decide on which requirements (change requests) to implement, considering the cost and impact.

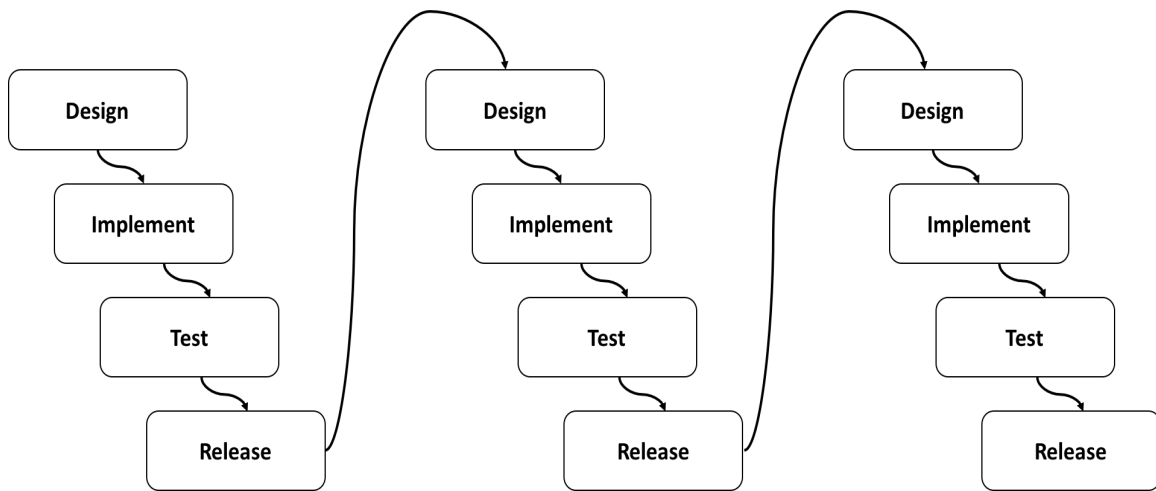


Figure 2.1: Figure showing Iterative software development

### 2.1.2 Software change

Software change can be defined as the process of altering software or constituting modules after delivery or release. This is primarily done to rectify faults, improve performance or adapting to changes due to internal or external factors [41]. The underlying reason for the change might originate from alternations in software landscape. for example marketplace, demand, stakeholders, technology etc. [26, 66]. Reacting and changing to these scenario is important because it helps to keep the software operational and relevant [56]. In order to accommodate the altering landscapes, software systems usually go through small and frequency changes. Software change is therefore an inevitable continuous process and there is a high probability that these changes will have adverse effects on the original software system [57].

Modern tools such as *Zendesk*<sup>1</sup> and other software project management technologies allow companies to collect user feedback related to their software product directly. They help to capture various changes to the shifting landscape. These feedback, in addition to regular followup client meetings and internal development monitoring by the software company themselves, generate many software requirement changes. These are converted to *Change Requests*, which are essentially software requirement specifications that define the requested changes in a thorough and systematic manner, outlining the user requirements and business needs [7]. Software changes generating these CR, start as soon as the first delivery of the software [22] is rolled out. The changes occur over the course of system development and maintenance [59].

Making changes to software has become particularly challenging because modern software systems are typically large and complex. In this scenario, software systems must be able to accommodate continual change to maintain utility and effectiveness [60]. Due to the sheer size of real-life software systems, it is impossible to identify and handle related changes manually. Key decision makers need to carefully understand and assess the risks before the design and implementation of change are initiated [19]. To accommodate and facilitate software changes, software developers must be provided with adequate tools [92].

Change requests are usually stored in repositories for future access and record keeping. As more than 85% of software requirements appear in the form of *Natural Language* [76], each CR is described with a long textual elaboration called *Description* and a subject referred as *Title*. The subject of these CR cover a wide variety of development aspects for example bug reports, improvements, feature requests, compatibility issues and much more. The primary challenge is to infer file impact with effort and duration requirements from words and phrases presents in the natural language representation.

---

<sup>1</sup>[www.zendesk.com](http://www.zendesk.com)

### 2.1.3 Vector Change Impact Analysis (VCIA)

Implementing software change requests consume valuable resources, primarily in the form of time and human resources e.g., analysts, developers, QA, testers etc. At any point in time, there is a backlog of change requests waiting to be implemented. As software development takes place in constrained environments, naturally there are not enough resources to implement all CR together. The problem is magnified for start-up and small MVP development focused organizations, operating with limited resources as well as additional time pressure. Planning for a release, the main objective of decision makers is to select CR subset which would provide maximum return on investment. In this case, investment being the allocation of time and the effort that personnel can put into the change requests. A crucial part of this decision making therefore need to take into account the required effort and duration requirement for implementing the change request. This will help to efficiently allocate the time and available effort on those CR which would return higher value and benefits for the organization.

An important point to consider here is that effort and duration, which are both measured in terms of time, is not the same. Effort determines the amount of work that will be required to complete a change request. It is usually measured in terms of man-hours. Duration, on the other hand, denotes the amount of time that will be required to complete the change request, which is usually measured in calendar days. The duration required is primarily determined by the perceived benefit. If the perceived benefit is high or if a change request address key functionality that requires immediate focus, then the CR will be worked on immediately. This is supported by multiple literature which report that the effort may not always directly correlate to the duration needed to complete a change request [27, 64]. Thus in thesis effort and duration are treated as different entities and predicted separately.

As traditional change impact analysis only looks at affected code components, with respect to any change, it is difficult to assess maximum return with only this information. Thus, the approach taken by traditional impact analysis which predict impact on the code base only, is not enough.

This thesis refers to these three dimensions of change request i.e., code files impacted, effort and duration as the *vector impact of change request*. In this thesis, three techniques for vector impact prediction are designed, proposed, implemented and validated. The proposed approach is called *Vector Change Impact Analysis (VCIA)*.

#### 2.1.4 Mining Software Repository (MSR)

Software repository store vast volumes of different software artifacts which are generated throughout the software development process. Modern repository archive, manage and maintains different data, starting with requirement data, followed by design artifacts, use cases, test cases, customer correspondence, evaluations and much more. Software repository provide a wealth of knowledge about various software engineering being. They are utilized for improving development and overall management. Some of the most popular applications include software effort prediction, task assignment, test management, sentiment analysis, theme analysis change impact analysis and more [22]. Mining Software Repository (MSR) refer to the wide class of tools and techniques which examine repository artifacts [44].

One of the most widely used repository is issue tracking system such as Jira <sup>2</sup> which hold various attributes of requirements data. Another popular repository used by almost all software companies is the version control system such as GitHub <sup>3</sup>, which in addition to maintaining various code versions, provide further insight such as code churn, developer activities etc. Information from these popular repositories are widely used for numerous software engineering research such as developer assignment [47], release readiness [5], requirement duplication analysis [91], theme/concept detection [37] and many more.

Another popular research direction is impact analysis, which is directly related to this thesis. As software change is initiated with a Change Request (CR) it is a common practice to analyze the repository, to find similar CR from the past and provide an prediction on the files impacted, effort

---

<sup>2</sup><https://www.atlassian.com/software/jira>

<sup>3</sup><https://github.com/>



and duration required. This is usually done by identifying and analyzing the code entities changed by these similar CR. This thesis follow a similar technique for evaluating the impact of CR. The objective of this thesis is to develop an aggregated technique which will take as input a change request. The output will be a prediction of the vector impact of the change request.

## 2.2 Literature Review

This section is dedicated towards reviewing the existing techniques related to this thesis. The segment starts with an overview, followed by a detailed discussion of each technique based on the in-depth analysis. This later part of the review is focused on exploration of the current state of evaluation and tool support of related studies. This segment of the thesis has been designed towards providing answers for RO1 and subsequent sub-objectives.

### 2.2.1 Existing techniques: code impact, effort and duration prediction

This segment analyzes existing literature on code impact, effort and duration prediction techniques. The review is separated into two parts. The first part discusses code impact prediction techniques, while the second part pertains to effort and duration prediction literature.

#### **Code impact prediction**

Most code impact prediction techniques are referred in the literature as change impact analysis (CIA) techniques. They refer to the collective techniques for identifying affected code components in response to proposed changes [9]. Another group of related literature is collectively referred as Feature Localization Techniques (FLT). FLT is a more simplified variant of CIA, where the objective is to identify the initial change entity, in response to CR. Some literature also associates the term *ripple effect analysis* to this group of studies. In this thesis, all these techniques related to identifying code change entities in response to change or CR, are collectively referred to as *code impact prediction* techniques. Code impact prediction can help developers and practitioners

Table 2.1: Existing literature related to code impact prediction. Each literature is described by: Technique (approach used for impact prediction, IR = Information Retrieval, CO = File co-change), Data source (data used for processing), Effort indicator(whether effort can be predicted), Duration indicator(whether duration can be predicted) and Industry case study (whether literature has been validated with industrial data).

Literature	Technique	Data source		Effort indicator	Duration indicator	Industry case study
		CR	Code			
Borg et al. [16]	IR	Yes	N/A	No	No	Yes
Gethers et al. [32]	IR, CO	No	Yes	No	No	No
Kagdi et al. [45]	IR, CO	No	Yes	No	No	No
Zanjani et al. [97]	IR	Yes	Yes	No	No	No
Canfora et al. [21]	IR	Yes	Yes	No	No	No
Canfora et al. [20]	IR	Yes	Yes	No	No	No
Torchiano et al. [93]	IR	No	Yes	No	No	No
Zimmermann et al. [99]	CO	No	Yes	No	No	No
Chochlov et al. [24]	IR	No	Yes	No	No	No
Zhang et al. [98]	LDA, IR	No	Yes	No	No	No
Lapeña et al. [53]	IR	Yes	Yes	No	No	Yes
Scanniello et al. [85]	IR	No	Yes	No	No	No

identify affected components, program comprehension, evaluate ripple effects, software testing and regression analysis [57].

There are many existing techniques for code file impact prediction. They span a broad spectrum of approaches, such as program slicing, graph-based approach [1], Bayes approaches [3], coupling techniques [33], metric-based approaches [63] and Information Retrieval (IR) based techniques. The methods can be broadly classified into two categories: static and dynamic impact analysis. Dynamic methods evaluate changes by analyzing various program execution data (execution trace, coverage, other execution data). Static methods on the other hand study semantic or evolutionary dependencies from existing or past system, to identify change propagation and affected components. This thesis focuses on static techniques, which utilize information stored in code and CR repositories and therefore relies on mining software repositories. Table 2.1 list code impact literature related to this thesis.

Gethers et al. [32] proposed an integrated approach to impact analysis consisting of three components. In the first component, textual descriptions from CRs is used by an IR method. This

relation is referred to as conceptual similarity or coupling, where there is a relationship or similarity between the concepts (requirements) and software entities (methods, files, classes, etc.). An intermediate representation is construed at the method level granularity by analyzing a single version of the code. Document descriptors are constructed using textual content of the methods from source codes. Based on the CR textual description, a query is run on the corpus to generate a ranked list impacted methods. The second component FPM uses past software repository data to analyze methods which have co-changed in earlier versions of the software. FPM generate rules which are subsequently used for making possible impact sets. Based on the analysis of past evolution of the software system, future changes are predicted. As the co-change or evolution are studied, these relationships are referred to as evolutionary coupling or similarity. In the third component, execution traces are generated with respect to individual files or features, representing a hierarchy of executed methods. Subsequently, IR method defined in the first element is applied only on the results of the execution trace. Different combinations of the three defined methods are integrated and the final impact sets are generated by taking different cutoff from the results of the constituting components.

Study by Kagdi et al. [45] combines conceptual and evolutionary techniques for CIA which supports file and method level granularity. The article is by the same group of authors as [32], the approaches are similar. This work is different concerning the exclusion of execution trace data. Also, the initial entity chosen for conceptual or evolutionary coupling does not come from textual bug descriptions or CRs. The selection of the initial entity will likely come from feature location activity. Based on an initially chosen entity (method or file) a ranked list of entities is generated based on conceptual similarity. Similar to [32] evolutionary couplings are made by applying FPM on software version data. Based on running experiments on open source data, the authors present statistical evidence showing that CIA is improved if conceptual and evolutionary approaches are combined for both file and method level granularity.

Canfora et al. [20] use past change data of multiple version, from software repositories to sup-

port IA. The proposed approach produces corpora containing revision comments and CR descriptions. The traceability between bug descriptions and revision comments from multiple versions of the development files are used for this method. When a new CR appears, the corpora will be queried to retrieve a list of the related files which are likely to be impacted by the proposed change. A ranked list of source files is generated based on textual similarity which is calculated using a probabilistic TF-IDF measure. TF-IDF works on the TDM and rates documents with respect to the query. The documents are ranked based on the occurrence of the critical query terms in the document representations, which emphasize terms that have more significant discerning capabilities. The appearance of textual descriptors in the new CR descriptions are matched to different files in the corpora. Those satisfying a minimum threshold are presented as a ranked list.

Authors Canfora et al. [21], propose an extension of the study done in [20]. In this research, a finer level of granularity is considered (i.e., line of code, file) and they are compared among themselves. In the file level, document descriptors are constructed by concatenating change request descriptions (both long and short versions) and the commit messages retrieved from past changes in the software repositories. At the line level, document descriptors are constructed using bug descriptions (again long and short descriptions) of the bugs and the commit comment from the past revisions. The bug descriptions and comments are retrieved by mapping line additions and deletions to bug entries as well as to the commit in the repositories. The operating procedure is the same as the previous approach [20], however the novelty is method of tracking line changes. Utilizing repository diff commands, specific line additions and deletion are monitored for each of the revisions.

The technique adopted by Zanjani et al. [97], combine interaction information with version history to derive an IR model. Interaction history represents file which have been interacted by developers but not committed. These files also play an essential role, because their interaction with relation to a bug fix represents the inherent influence and impact on other files. In this approach the corpora are created, where each row of the TDM is a revision which is composed of comments and

identifiers from interacted and committed source files, commit messages of the revision and associated bug descriptions. Data dimension is reduced with SVD and projected onto an n-dimensional space of the KNN model. Cosine similarity distance is measured to evaluate closeness of incoming queries with respect to documents in the corpus.

Torchiano et al. [93], use an IR technique for change resolution. The proposed approach constructs corpora of file descriptors where each entity is described by a combination of comments in the source code and commit messages. The queries are generated using combinations of terms appearing in CR descriptions. Ranking of the files are created based on the occurrence of query terms in the corresponding file comment or commit message data. The proposed approach was verified on four open-source projects.

The technique outlined by Zimmermann et al. [99], is perhaps one of the most cited and influential papers in this domain. The paper outlines the FPM on historical software data. The proposed approach works at the file as well fine levels of granularity. Applying the Apriori algorithm on past changes transcribed as transactions of entities, the proposed approach is capable of identifying impact sets at the different level of granularity. Based on the working methodology a tool was developed which can show developers impacted items as they are making changes to the code. The method is applied on eight open-source projects and shows good Precision and Recall values.

The following segment discuss key findings of code impact prediction prediction literature review related to RO1.1

- **Decision indicator (effort & duration):** From the Table 2.1 it is clear that none of the mentioned literature perform effort and duration prediction. These techniques are focused on generating code impact prediction. The initiating entity can be a file, method or class, based on which code impact is generated. The effort or duration required to implement the change is not considered by any of these approaches. Developers and decision makers would need to rely on their gut feelings and instincts to make these predictions.

- Data source:** All the literature listed utilize two primary data sources i.e., *CR* and *Code*. *CR* refers to any data related to change requests such as title, description etc. *Code* data on the other hand refer to information such as code contents (comments and identifier names), commit data and change-logs. As software development is a continuous process, code files would be regularly updated, which in turn would cause change to code content and other attributes utilized in TDM. As a result, the corresponding corpus would need update every time there is any code change. *CR* data source on the other hand is much more stable, compared to code data. *CR* do not go through continuous and massive changes. They usually go thorough minor updates to status and assignment attributes. This in turn means that the corpus will require very little change to stored *CR* attributes.
- Techniques:** All mentioned literature use IR for code impact prediction, which refer to retrieving relevant information from an corpus or knowledge base. In all the literature, a corpus is created using data retrieved from different repositories. The corpus is usually represented in terms of a term document matrix (TDM), which stores key terms related to each of the entities (i.e. documents). Entities can be files, methods, class, attributes etc, data related to which are extracted and stored. After the corpus is built, incoming changes are formatted into queries that conforms to the corpus structure. Based on similarity of the the query with the TDM entities, impacted entities are ranked and listed. Similarity is defined using TF-IDF and cosine similarity (explained fully in chapter 3) Another popular technique is utilization of file co-change (CO) patterns. It is based on the popular Apriori algorithm [4]. The main idea is to organize software entities which have co-changed together are transactions, based on which rules are generated using historical entities which have changing together (co-changed).

## Effort & duration prediction

Table 2.2 lists the literature related to effort and duration prediction, related to this thesis. All literature shown in Table 2.2 make duration or effort prediction based on CR. Depending on the technique used, different CR attributes are taken as input. The output is either duration or effort prediction.

*Issue Resolution Time (IRT)* prediction refers to the class of techniques whose objective is to predict the duration or time required for a CR (issue) to close. The duration is the difference between the opening and closing date of an issue. In this thesis, IRT prediction is referred to as *duration* prediction. Some literature refer IRT as *issue closing time*, *issue life-time*, *issue/bug fix time* and *issue resolution time*. Duration prediction is important because it helps with efficient planning, scheduling and project organization [72]. The following six studies relate to duration prediction [2, 10, 34, 52, 64, 72].

*Effort* prediction studies aims at anticipating the amount of effort that will be required to implement change request. There are few studies which attempt effort prediction from change requests. The following two studies [27, 94] refer to effort prediction, with respect to CR. Effort is usually measured in person hours. When new change requests are created, an initial prediction of required effort are entered into issue repository such as Jira. It represents the number of hours that developers might need to spend for completing the CR.

Kikas et al. [52] utilize various CR attributes for making issue duration prediction. Random Forest [17] classifier is used to train the duration model. Both static and changing CR attributes are included into the model. The attributes relates to submitter, participant, project and textual descriptions. Performance observed for different observation points (number of days for which issue have already been open). Experimentation revealed that accuracy improved when trained with one day to one week old issues. Another similar study [64], also utilizes Random Forest classifier but on different set of attributes e.g., location, reporter and description. The focus here was on bug duration prediction. Their conclusion was that attributes established by earlier studies

as important, have little impact on the CR duration.

Pfahl et al. [72] study and apply existing IRT prediction techniques in an industrial context. The aim of this paper was to understand the extent to which expert duration predictions could be complemented and improved with automated techniques. Experimental analysis revealed that textual analysis was able to improve expert duration prediction. Assar et al. [10] is a replication study, which attempt to repeat the experiments of an earlier bug resolution time study done by Raja et al. [75]. In addition, the performance of automatic clustering for duration prediction is evaluated. Clustering of CR was done with RapidMiner<sup>4</sup> tool. Experiments were run with different subsets of the data. The researchers concluded that automated clustering techniques cannot perform duration prediction with satisfactory levels of accuracy. The last literature on bug duration prediction Giger et al. [34], which use Decision Tree classifier on two class of CR attributes i.e., *Constant* and *Changing* attributes. Experimenting with OSS data, they conclude that some post submission attributes improve prediction accuracy.

This segment discuss on effort prediction studies. Dehghan et al. [27] propose a hybrid combination of textual and classification techniques for effort prediction. The objective is to make prediction for CR type *Tasks*. For the classifier model, both textual and non-textual CR attributes are utilized. The classifier model is stacked on the textual similarity model, although the exact details are unclear. The proposed approach was applied on two industrial data. They concluded that some attributes were important for making prediction in their industrial context and those were highlighted. Another work by Weiss et al. [94] propose technique for predicting the effort required to fix bugs. With respect to a new change request, the approach defined in the paper find similar change requests from past history. The predicted effort of new change request, is the average of similar change requests. Experimentation and analysis on OSS data reveal that performance is good if sufficient historical cases are available.

The following segment discuss key findings of effort and duration prediction literature review, towards fulfillment of objective RO1.1:

---

<sup>4</sup><https://rapidminer.com/>



- **Decision indicator (code impact):** All the techniques perform effort or duration prediction, using different change request attributes. However, these techniques are not integrated with any code impact prediction techniques. The techniques mentioned do not indicate the files that might be impacted with respect to change request.
- **Type:** The type indicates the type of CRs for which the corresponding technique generates prediction. Not all types are supported by either effort or duration prediction literature. For existing literature on duration prediction, all CR type is supported by only two studies [52, 72], while the remaining make predictions for *Bug* type CR. On the other hand, for effort prediction none of the literature supports all CR type. In fact, one study supports *Bug*, while the other supports non CR entity called Work Item whose type is *Task*.
- **Technique:** Duration and effort prediction techniques can be categorized into two groups: textual analysis and classification techniques. Textual analysis uses CR text attributes, which are primarily *Title* and *Description*. Subsequently textual similarity or clustering is used to generate predictions. Classification techniques, on the other hand, extract various CR attributes and train classifiers to generate effort prediction. The attributes cover all aspects of CR, some of which are: submitter, assigner, developer, discussion, date and time.

### 2.2.2 Analogy based problem solving

Much of human cognitive reasoning and decision making is defined by how similar problems were resolved in the past. This is referred to as reasoning Analogy Based Reasoning (ABR) [90]. The idea is to derive solutions by linking information from past solution instances. ABR contains a knowledge base, which contains all cases of available prior solutions. For a new problem instance, similar problems are searched and retrieved from the knowledge base. The corresponding similar

Table 2.2: Existing literature related to effort and duration prediction. Each literature is described by: Technique (approach used to make prediction, KNN = K Nearest Neighbour), Type(the types of CR supported), Code Impact Indicator(whether code impact can be predicted) and Industry case study (whether literature has been validated with industrial data).

Literature	Technique	Type	Code Impact Indicator	Industry Case Study
Kikas et al. [52]	Classifier	Any	No	No
Pfahl et al. [72]	Classifier, KNN, K-means	Any	No	Yes
Assar et al. [10]	Classifier	Bug	No	No
Abdelmoez et al. [2]	Classifier	Bug	No	No
Marks et al. [64]	Classifier	Bug	No	No
Giger et al. [34]	Classifier	Bug	No	No
Dehghan et al. [27]	Classifier, K-Means	Work item-Task	No	Yes
Weiss et al. [94]	KNN	Bug	No	No

solutions are extracted, analyzed and adapted to derive the final solution. The following are define the key stages of ABR [36]:

- **Knowledge base representation:** This is the initial stage of ABR, where past problems instances are stored. For each problem, the corresponding solution(s) applied are also resolved and stored, to be used for prediction. Together the problem and solutions represent the knowledge base against which new problems will be compared for generating solutions.
- **Search and retrieval** Based on the knowledge base created in the previous stage, searching is performed in this stage. The searching is performed with respect to the query problem, for which a solution need to be generated. Searching is performed on the entire knowledge base, where the analogy or similarity between the query and existing problems are defined by a similarity measure.
- **Selection:** Depending on the similarity values between the query and existing problems, the query problem is mapped on the existing problems. A threshold value determines the mapping between the query and existing problems. The result of the

mapping is a list of existing problems (candidate problems) which are analogous (similar) to the query problem.

- **Prediction:** From the list of analogous problems with respect to the query, the solution is generated for the query problem. This is done by analyzing solutions of the candidate problems. The existing solutions are adapted to generate a final prediction for the query problem.

Analogy based prediction are popular because they are capable of capturing complex relationships between the target concepts and software properties [40]. Using analogy allows solutions to be derived based on actual events or solutions rather than using theorized and assumed models [87]. In addition, analogy allow decision makers to see the solution deriving process, which makes solutions more acceptable as they are intuitive and derived from actual past solution [40,87].

Analogy has been applied and used in various segments of software engineering dimensions [48] including software quality, effort prediction, reuse and management among others. In [49] analogy was used to predict software quality as either faulty or non-faulty by retrieving similar modules from history and analyzing defect counts. A related study [51], predicts the number of faults in a system. Quality in terms of risk has been studied by related literature to classify software into risk categories [50]. Software reuse is another area where analogy based prediction have become studied. Morris and Mitchell [68] recommend software artifacts be reused from current software. Another study by Gonzalez et al. [35] enables reuse of classes from current software. Analogy based recommendation have also been used to recommend various software artifacts i.e., user interface components and software design artifacts among others [48].

The set of literature which relates most to this thesis belongs to software effort prediction category. The work by Finnie et al. [30] is a popular literature where software effort prediction is performed utilizing analogy. Over the years many researchers have enhanced their respective studies with respect to analogy based software effort prediction. Idri et al. [39] combine analogy with other machine learning techniques to form a hybrid framework for effort prediction. Li et

al. [58] propose various methods to handle data irregularities such as missing or irregular data points. A similar study by Mendes et al. [67] use analogy based effort prediction in web projects. All the effort prediction studies mentioned here are designed and built for effort estimating datasets, which represent software size as a function of different project characteristics.

Software effort prediction has been historically viewed as a function of size. The effort is calculated in terms of man-hours, or alternatively in similar time units such as man-days/weeks/months [31]. Size is represented in terms of different software metric. Function Points is one of the popular software size metrics which express the amount of functionality provided by a software product [6]. Using Function Point analysis (FPA) numerous literature have attempted to predict the effort of software projects by looking at different data sources. Another popular approach is to use software Lines of Code (LOC) as size metric. LOC is utilized by the popular cost prediction model COCOMO [15]. Both FPA and COCOMO require computing key software variables in order to obtain effort predictions. FPA require defining five key parameters i.e., external input, external output, external inquiry, internal logic file and external interface file. On the other hand COCOMO require determination of adjustment factors with cost drivers. These models were originally designed for linear models such as waterfall where entire software project parameters are defined and known beforehand. However iterative software development has gained great popularity, where everything is not defined upfront and each iteration may potentially proceed in totally different directions. Development plan and goals change very frequently. All these make definitions of key variable very difficult [38] in an iterative development methodology such as Scrum. Studies such as [83] and [74] have tried to see the effect of applying the original techniques in an Agile (iterative) setting and recommend modifications to make these effort prediction techniques better suited to iterative development.

However, these techniques still require project parameters and cost drivers to be defined. Thus the accuracy with which these variable need to defined depends on experience and knowledge of the decision makers. In order to get accurate prediction require development processes to be estab-

lished and clearly defined. Finally, defining the key variables require calculations and processing which utilize valuable man-hours which developers can spend doing actual development work. For small and start-up companies this does not make sense because a lot of the development process is not defined, which would make some variable setup such as cost drivers, very difficult. Also, these small organizations have limitations in man-hours and are therefore less likely to invest scarce human resource towards necessary calculations and analysis required to make effort estimations based on functions points or lines of code.

This thesis utilizes ABR for determining file impact, effort and duration prediction of a change request. For a new change request, the textual similarity is used to define analogy between change requests. As mentioned previously existing impact analysis techniques only make code impact recommendations. The goal here was to improve the impact analysis decision making by augmenting the file impact prediction with that of effort and duration. Using textual similarity file impact could be predicted. The effort and duration prediction is derived from the same analogy and essentially provide effort and duration prediction at zero or very minimal cost, as because no new data need to be analyzed. The objective here was to provide vector impact prediction utilizing minimum investment in terms of data preparation and human invention. The focus here was not on proposing a new effort or duration prediction technique independently. Existing analogy based effort prediction techniques (described in this section) rely on various project data attributes. The effort is predicted by measuring the similarity between these attributes. However, these project attributes are not recorded for small and start-up companies. The approach outlined in this thesis can generate an effort prediction even for that scenario when the data (of different project attributes) is not available. The prediction needs are made only considering the change request illustration i.e., summary and description because change request repository is widely employed by companies of almost all sizes [22]. Companies use change request repositories because it makes planning and change management of software much easier.

In this thesis, the knowledge base is constructed using data from change request and code

repository. For each available past CR data, their description and title are stored in the knowledge base, which serves as problem definition. The corresponding files impacted, the original effort predicted and duration taken, are stored as solutions (against CR description and summary). Each of the ABR components with respect to this thesis is defined in the next chapter.

### 2.2.3 State of industry evaluation

Collaboration between industry and academia is always challenging. Companies are reluctant to share data and information due to many reasons, primarily due to privacy and security concerns. The other key reason is the lack of interest towards research and collaboration. This trend is also true for the literature related to this thesis. From Table 2.1 and 2.2, only four studies in total have any kind of industrial validation i.e., Pfahl al. [72], Dehghan et al. [27], Lapeña et al. [53] and Borg et al. [16]. Furthermore, only Pfahl al. [72] literature uses both industrial and OSS data for evaluation. The remaining studies only report performance on OSS data. Because industry and OSS projects are different regarding policies, procedures, tools, and techniques comparing with one set of studies is not enough. To ensure that results are repeatable and transferable [71], evaluation should complete on both OSS and industry data with appropriate comparisons. These findings are related to fulfilling objective RO1.2.

### 2.2.4 State of tool support

Table 2.3 lists the relevant tools available for impact analysis. The table also show functionality that each tool offers. The criteria for deciding the functionality is primarily based on studying the status of tool support mentioned in [57]. Another key input for deciding functionality was understanding needs of software company Brightsquid Inc. These findings are related to fulfilling objective RO1.3. The following discuss existing tools with respect to the functionality offered:

- **Language independence:** With the exception of FLOrIDA [8], all tools are language dependent, it built for and tested on Java. This restricts applicability to other

language platforms and environments, especially with the emergence of popular platforms/languages such as Python, C#, Ruby etc.

- **Industrial validation:** FLOrIDA [8] and Chianti [77] possess industrial validation. Industrial validation is important because it will ensure that results are repeatable, robust and transferable.
- **IDE dependence:** All the tools except FLOrIDA [8] are built as an IDE plugin for Eclipse. This means that developers would need to use a specific IDE, which might prove counter-productive because everyone has their own preference and comfort zone.
- **Textual query:** Textual input is important because change requests are usually formulated in natural language. In addition, without textual query input, if change impact can only be measured with respect to *method*, *class*, *file* inputs, then only developers or technical people will be able to benefit from this tool. FLAT3 [84] and ImpactMiner [28] accept textual query as input.
- **Human intervention:** Only two out of the five tools can operate without human intervention. Operating without human intervention is important for two reasons. The first is the ability to work autonomously reduce chance of human error and secondly, tool can be used by anyone without any knowledge of the system.
- **Effort & duration indication:** None of the tools provide support for effort and duration prediction. Their main focus was the prediction of code impact. Thus it would be very difficult to make decisions which would maximize return in investment.
- **Code impact indication:** All the tools are able to perform code impact predictions.

Table 2.3: Tools for code impact analysis, where each tool is evaluated against multiple functionality defined on the left.

Functionality	Tool				
	JRipples	FLAT3	ImpactMiner	FLOrIDA	Chianti
	[18]	[84]	[28]	[8]	[77]
<b>Language independence</b>	No	No	No	Yes	No
<b>Industrial validation</b>	No	No	No	Yes	Yes
<b>IDE dependency</b>	Yes	Yes	Yes	No	Yes
<b>Textual query</b>	No	Yes	Yes	No	No
<b>Human intervention</b>	No	Yes	Yes	Yes	No
<b>Code impact indication</b>	Yes	Yes	Yes	Yes	Yes
<b>Effort &amp; duration indication</b>	No	No	No	No	No

## 2.3 Summary

This chapter presents the background and fundamental concepts related to this thesis. Existing literature related to code impact, duration and effort prediction techniques are reviewed extensively following objective RO1. Analysis show existing literature to be focused on the code impact of change. There is no existing technique which makes code impact, effort and duration predictions simultaneously. From preceding discussion, it is clear that effort and duration requirement with the code impact (vector impact) is vital for effective decision-making. Further examination of performance evaluation, highlight the need for more industrial evaluation. Finally, investigation of tool support outlines the need for the development of generic tools which can provide effort, duration, and code impact prediction. The findings meet the goals defined by objectives RO1.1, 1.2 and 1.3



## Chapter 3

### **Proposed approach: Vector Change Impact Analysis (VCIA)**

This chapter describe the proposed techniques for predicting vector impact of a change request. First segment begin with description of the Analogy Based Reasoning (ABR) process for the proposed approach. Each of the core components of ABR are discussed in details, including discussions on data utilized, pre-processing steps, similarity measures, process overview and lastly process/algorithm of the three proposed techniques.

Some contents in this chapter come from the paper [43], which reports the application of VCIA in Brightsquad. In particular the algorithms, related processing, the process outline and evaluation measures come from the mentioned paper. In most cases some modifications were made to the paper components before they were placed in this thesis. The evaluation measures for effort and duration prediction used in this thesis is different. The components from the original paper (with or without modifications) are cited with reference to the paper, as they share the same core essence. This includes tables and figures from the paper [43]. For most of the cases, the description in this thesis provide a more detailed account compared to the version in the paper.

#### 3.1 Vector Change Impact Analysis (VCIA)

The proposed approach of this study predict the vector impact of software change request. Subsequently, the proposed technique is titled **Vector Change Impact Analysis (VCIA)**. The overall VCIA experimentation and evaluation process appear in Figure 3.1. Every step, beginning with data acquisition and ending with result analysis are mentioned in the outlined process. Each component is discussed in the subsequent sections. This thesis analyzes the performance of three techniques for different components of the vector impact prediction i.e., files impacted, effort and duration required.(step 4 of Figure 3.1). Experimental analysis revealed (fully detailed in the next

chapter) that usage of these (three) techniques provide better prediction performance for different components of the vector impact prediction i.e., effort, duration or file impact. Thus all the three separate techniques are presented, and they are all defined under the single umbrella of the aggregated proposed technique titled VCIA, thus the term “An” is used at the beginning of the title. Furthermore as all the (three) techniques rely on analogy for making respective predictions, the term “analogy” is part of the thesis title. The following subsections defines each of these techniques separately. The process outline and the techniques appear in the paper [43]. The paper version provides the main essence, while the contents of this chapter provide a more detailed description. All the techniques follow the Analogy Based Reasoning (ABR) process. The following sub-sections define each of the major ABR components, which were introduced and discussed briefly in chapter 2.

### 3.1.1 Knowledge base representation

The knowledge base is defined as  $KB = \langle CR_M, P, V \rangle$ .  $CR_M = \{cr_1, cr_2, cr_3, \dots, cr_m\}$  is the set of change requests  $cr_i$  (extracted from repository history mining).  $P$  represent the set of attributes describing each change request, where  $P = CRC \cup CRT \cup \{Effort, Duration, File\}$ . Here,  $CRC$  represent the processed and concatenated textual content of CR Summary and Description (data source explained in the subsequent parts). Similarly,  $CRT$  represent the topic derived by applying Latent Dirichlet Allocation (LDA) on CR Summary and Description. Topic are refereed to as *topic term* or *term topic* in the algorithm descriptions in the later part of this chapter. Here,  $CRC(cr_i)$  and  $CRT(cr_i)$  signify processed textual content and LDA topic respectively of change request  $cr_i$ . LDA topic is only used in BOW & LDA algorithm discussed later.  $Effort(cr_i)$ ,  $Duration(cr_i)$ ,  $File(cr_i)$  represent effort, duration and file(s) impacted respectively, of change request  $cr_i$ . Finally  $V = \{P(cr_i)\}$  represent the domain of attribute values ( $CRC$  and  $CRT$ ) for all change requests  $CR_M$ . Here,  $CRC$  and  $CRT$  represent the problem which are used to predict the solutions: file(s) impacted, effort and duration of a change request. The following sub-section describe the data usage, textual pre-processing steps and solution value resolution for building the  $KB$ .

### *Solution resolution - files impacted, effort and duration*

This sub-section describes how the solution attribute values (defined in KB representation) are derived from CR and code repository historical data.

- *Files impacted*: Files impacted refers to the files which have been modified by any change request. The code repository record all modifications to various code files, where changes are batched together as *commits*. As a practice, developers link commits to CRs, by mentioning the CR identifier in commit messages. These links help to identify the requirements causing code changes. The relationships between CR and files changed (in GitHub) are identified by analyzing the presence of CR identification (ID) number in commit messages. The process is formally known as traceability establishment [89]. This has been achieved in this thesis with multiple regular expressions. Step 2 of the process outlined in Figure 3.1, represents this linking process.
- *Effort*: Effort refers to the effort value (estimated) predicted and entered into the CR repository when new requests are created. The values are entered in units of person-hours, but this thesis use the second converted value of effort, because web interface data access converts the data to this unit.
- *Duration*: Duration refers to the number of the days it takes to close a change request. It is calculated as the difference (measured in number of days) between the opening date and closing date.

### *Data usage*

This sub-section highlights the data attributes used from the various repository to prepare the knowledge base (defined in the earlier sub-section). VCIA utilize data from two repositories; change request data from JIRA issue management system (change request (CR) repository) and code data from GitHub version control system (code repository). Table 3.1 shows the main data

attributes being utilized. The *Repository* column specifies the data source i.e., CR repository data or code repository. *Usage* column define purpose of data usage. Column *Type* denotes the type of data e.g., numerical, text etc. Finally, *Availability* describes to what extent the data is available in the repository.

Table 3.1: Data attributes from CR and code repositories utilized by VCIA. For each attribute the *Name*, *Repository*, *Type* and *Availability* are shown.

<b>Attribute Name</b>	<b>Repository</b>	<b>Type</b>	<b>Usage</b>	<b>Availability</b>
CR Summary	CR	Text	Similarity	High
CR Description	CR	Text	Similarity	High
Original Effort Estimate	CR	Numeric	Effort estimation	Moderate
Open date	CR	Date	Duration resolution	High
Close date	CR	Date	Duration resolution	High
CR ID	CR	Text	Traceability	High
Commit message	Code	Text	Traceability	High
Commit files changed	Code	Text	Files changed resolution	High

### *Data pre-processing*

This segment describes the data pre-processing, which has been formulated based on experimentation, related literature and techniques mentioned in surveys such as Chen et al. [23]. Data-preprocessing defines how the textual summary and description data are processed. The pre-processing has been designed to make the data easily and readily analyzable by the different methods. The pre-processing steps applied on the data of this thesis are identical to those reported in the paper [43]. The end result is a concatenated version of the summary and description after the following pre-processing steps have been applied:

- *Basic text cleaning*: The first step is text cleaning where special characters, code snippets, hyperlinks, email addresses and other elements not pertinent to the analy-

sis are removed.

- *Removing Stop Words*: Commonly occurring words such as “and, is, are, the” are removed as they do not add any value to the analysis. The list has been put together from several pieces of literature and adding terms based on independent study.
- *Stemming*: Porter Stemming algorithm [73] is a lightweight and popular technique for reducing different morphological forms of the same words to their root state. For example *consign, consigning, consigned, consignment* are all reduced to the root form *consign*. Otherwise, different morphological forms of the same word are treated as unique occurrences, which adversely affects the textual analysis. In addition to stemming Lemmatization operation was also evaluated. However, the performance of stemming was slightly better (see figure A.6), therefore it was chosen for all experiments.

### 3.1.2 Search and retrieval

Let  $CR_q$  be a new change request for which vector impact prediction is required. With respect to ABR process,  $CR_q$  here represent a new problem for which a solution need to be formulated. With respect to the query, the knowledge base would be searched and relevant results need to be ranked and presented. For searching the knowledge base and providing a ranked list of results Cosine similarity measure is used in this thesis. Cosine similarity is one of the most popular techniques for measuring the textual similarity between two documents [11]. This measure has been applied for searching and retrieving relevant instances for many applications including change impact analysis. The same similarity measure was also used in [43]. Similarity (analogy) between two change requests is calculated using Cosine Similarity of their textual content. TF-IDF (term frequency-inverse term frequency) is a statistical measure used to evaluate the importance of words in a collection of documents [62]. It consists of two components, *Term Frequency (TF)* - count of the number of times a word appears in a document, which is then normalized by the total number

of words in that document. The second component is the *Inverse Document Frequency (IDF)* - logarithm of the number of documents in the corpus divided by the number of documents where the particular term appears. TF-IDF is calculated on *CRC* or *CRT* (cosine similarity is also used to measure term topic similarity in BOW & LDA.) attribute of change request  $CR_i$  and subsequently used to represent change requests in the vector space.

Representing each CR as a vector in vector space, The similarity  $S$  between CR is calculated as follows:

$$S(CR_q, CR_i) = \frac{\vec{V}(q) \cdot \vec{V}(d)}{|\vec{V}(q)| |\vec{V}(d)|} \quad (3.1)$$

where  $CR_q$  and  $CR_i$  represent query and one instance of existing CR respectively.  $\vec{V}(q) \cdot \vec{V}(d)$  represents the dot product of the weighted vectors with  $|\vec{V}(q)| |\vec{V}(d)|$  being their Euclidean norms.

### 3.1.3 Selection

The similarity measure provide quantitative method of ranking existing CR from the knowledge base. The value of the similarity define the level of similarity (analogy) between query  $CR_q$  and existing change requests  $cr_i$  in  $CR_M$ . Using the values obtained from the evaluation measures, top  $N$  similar change requests (or analogies) are selected from  $CR_M$ . Threshold  $T$  values are used to define the final  $N$  number of similar CR. The threshold values are discussed in the relevant sections.

### 3.1.4 Prediction

From the list of  $N$  selected analogy from the knowledge base, file impact, effort and duration are predicted. This part of ABR involve the step *Adaptation*, where existing retrieved solutions from top  $N$  analogies are used to make the required prediction for  $cr_q$ . For effort and duration, prediction value calculation is defined as follows:

$$PredictedValue(CR_q) = \frac{\sum_{i=1}^N value(i)}{N} \quad (3.2)$$

where  $value(i)$  is the duration or effort value of the  $cr_i$  change request. The calculation here is the mean over  $N$  effort or duration values coming most  $N$  most similar CR. Following experimental analysis and following recommendations of similar studies such as [58,67] other value calculation processes were tried and experimented (see figure A.7). Overall the mean returned best accuracy figures, it was therefore used for effort and duration value prediction. For file impact prediction, process of generating file impact list vary with the technique. The details are described in the subsequent sections with the algorithm description.

## 3.2 Vector impact prediction techniques: algorithm and description

### 3.2.1 Bag of Words (BOW)

The primary motivation of this model is to utilize similarity between change requests. Typically change requests discuss some specific aspect of software. It is common to have different change requests addressing similar or even the same elements of software. Consequently, their textual descriptions will also be similar. Therefore textual similarity can be used as a proxy of aspect affinity between change requests. Correspondingly, files impacted by similar requests are analyzed, it is common to find the same set of files recurrently changed. On the other hand, analysis of required effort and duration of similar requests reveal that they share similar or close values. Hence, change requests which are textually similar are also likely to change the similar set of files. At the same time, the effort and duration values are also expected to be similar, i.e., similar textual content translates to similar vector impact.

The base model of this thesis is Bag of Words (BOW) [62], a data representation technique, mainly used in natural language and information retrieval applications. BOW represents textual data as multi-set of constituent words, discarding the order or other attributes. The main idea of BOW is to construct a knowledge base consisting of existing changes. BOW also appear in [43].

Each CR is represented as a BOW entity consisting of corresponding CR Summary and Description. From the data retrieved from repositories the corpus is created that represents knowledge

base against which new change requests will be evaluated. Using this knowledge base, file impacts for new change requests are predicted following steps of Algorithm 1. The algorithm takes as input all change requests in the knowledge base and the query CR text (processed and concatenated summary and description),  $CR_M$  and  $CR_q$  respectively. The query CR represents the new CR, for which file impact needs to be predicted. The similarity is calculated between each query and existing CRs. Apache Lucene<sup>1</sup> is used to index and query the data entities for finding suitable matches, with respect to input CR.

### *Selection & Prediction*

If threshold  $T$  is satisfied, the corresponding CR is marked as similar. Similar requests  $SCR$  was first sorted according to  $CR_q$  textual similarity. In addition to sorting with respect to query CR textual similarity, sorting of files was also attempted with respect to decreasing order of *Total code churn* - number of lines added and deleted and decreasing order of *Days since last modified* - number of days ago files were changed. As sorting by decreasing order of textual similarity generated best results, it was used for subsequent analysis. From sorted  $SCR$ , corresponding *file changes*, *effort and duration* ( $sr_f, sr_e, sr_d$  respectively) are added to final prediction outputs i.e.,  $I_F, I_E, I_D$  respectively. Thus for a (new) query change request, the output is a list of impacted files, effort and duration, denoted by  $I_F, I_E, I_D$  respectively, sorted according to textual similarity.

### 3.2.2 Bag of Words & Latent Dirichlet Allocation (BOW & LDA)

Many techniques are available for identification of prominent textual concepts from textual data. Topic Modeling (TM) using Latent Dirichlet Allocation (LDA) [14] is one such approach which can be used to extract topics, i.e., concepts from CR textual data. Based on the topic - CR relationship, each CR can be assigned to topics and corresponding key topic terms. Analyzing topic terms and relevant vector impact (file, effort and duration), it is hypothesized that CR's with similar topic terms will have similar vector impact. However, one of the drawbacks of LDA is the complete un-

---

<sup>1</sup><https://lucene.apache.org/core/>



---

**Algorithm 1** Bag of Words for vector impact prediction [43]

---

**Input:**  $CRC_q, CR_M$  in**Output:**  $I_F, I_E, I_D$  out

```
1: for  $i = 1$  to  $M$  do
2:   Calculate  $S(CRC_q, CRC_i)$  {using equation 3.1}
3:   if  $(S > T_B)$  then
4:     Add  $CR_i$  to list of similar CRs  $SCR$ 
5:   end if
6:   Sort  $SCR$  according to similarity values w.r.t  $CR_q$ 
7:   for each element  $sr \in SCR$  do
8:     Add all  $f \in sr_f$  to  $I_F$ 
9:     Add  $sr_e$  to  $I_E$ 
10:    Add  $sr_d$  to  $I_D$ 
11:   end for
12: end for
13: return  $I_F, I_E, I_D$ 
```

---

supervised nature of topic extraction. In most cases some sort of supervision is beneficial towards extracting useful topics for subsequent analysis. This is inline with the findings of a separate study, where LDA topics were used for file impact prediction [69]. This study concluded that file impact prediction performance improves if human terms are integrated with machine terms (topics) and associated with CRs. But expert manpower was not available for introducing human terms to the large number of data points (CR) that needs to be analyzed for this thesis. The human terms (associated to CRs) would be used to compute similarity. But as the human terms were not available, textual similarity from BOW is integrated with topic (machine term) similarity to act as proxy for the missing human associated CR terms. This provides an additional direction of similarity calculation and reduces total reliance on machine topics for similarity calculation between change requests. The basic procedure of BOW & LDA is also reported in [43].

In this technique, LDA is applied on all available change request  $CR_M$  (pre-processed Summary and Description text) to generate  $Z_K$  topics, where membership of topic  $z_k \in Z_K$  for change request  $cr_i \in CR_M$  is  $\theta(cr_i, z_k), \forall i, k : 0 \leq \theta(cr_i, z_k) \leq 1$  and  $\forall i : \sum_i^k \theta(CR_i, z_k) = 1$ . Membership threshold was set at  $M_{TP} = 0.20$ , resulting each CR query  $cr_q$  belonging to at least one  $z_k$  of the  $K$  topics, where  $K = 20$ . Next, word topic membership vector  $\phi_K$  is analyzed for determining topics. For

each word in  $\phi_k$ , if  $\phi(\text{word}, z_k)$  is greater or equal to  $T_{TP} = 0.80$ , corresponding word is added as the topic terms  $z_k$ . Based on experimentation and reviewing literature such as [13], [23], the value of  $M_{TP}$  and  $T_{TP}$  have been fixed.

Each CR is considered as a separate document and LDA is applied on all available CR set  $CR_M$ . LDA extracts *topics* which are a collection of associated and connected *textual terms* appearing together in CR textual representations. Each term has a degree of association with the topic *topic-term probability* and the threshold  $T_{TP}$  determines if a term will be included/selected for the topic. Each *topic* will be defined by a set of textual terms (*topic terms*), which had a association probability greater than  $T_{TP}$ . After this, CR would be assigned to topics depending on the document-topic probability. If the *document-topic probability* is greater than  $M_{TP}$ , then corresponding *topic* is assigned to the CR. Consequently, each CR will be assigned to at least one LDA topic and each CR may be assigned to multiple topics (and associated terms). Finally, each CR is associated with *topic terms* of all *topics* to which it has been assigned. Algorithm 2 utilize the *topic terms* to find similarity and combine with BOW textual similarity defined in algorithm 1 to make vector impact predictions. To perform LDA, Mallet [65] tool implemented in Java has been used. The tool performs LDA and generates *document-topic probability* and *topic-term probability* matrix. Using these values, subsequent analysis and calculation are completed.

The algorithm takes as input all change requests, query CR text (summary and description) and query CR topic terms ( $CRT_M, CRC_q CRT_q$  respectively). Query CR represents the new request, for which file impact, effort, and duration needs to be predicted.

### *Selection & Prediction*

The similarity between query topic and each of CR assigned the topics terms is calculated. If threshold  $T_M$  is satisfied, all files from CRs with the similar topic terms are added to  $I_{TM}$  (Steps 4 to 7). File impact list is generated  $I_{BOW}$  using textual content similarity (same procedure as BOW). The final list of impacted files  $I_F$  is the intersection of  $I_{BOW}$  and  $I_{TM}$ . For Effort and duration prediction, intersection of CR  $ICR$ , coming from BOW ( $SCR$ ) and LDA ( $SL$ ) is generated.  $ICR$  is

---

**Algorithm 2** BOW & LDA similarity for vector impact prediction (adapted from [43])

---

**Input:**  $CRC_q, CR_M, CRT_q$ **Output:**  $I_F, I_E, I_D$ 

```
1: for  $i = 1$  to  $K$  do
2:   Calculate  $S(CRT_q, CRT_i)$  {using equation 3.1}
3:   if  $(S > T_M)$  then
4:     For each  $cr$  with similar topic terms, add to  $SL$ 
5:   end if
6:   for each  $sl \in SL$  do
7:     Add all  $f \in sl_f$  to  $I_{TM}$ 
8:   end for
9: end for
10: for  $i = 1$  to  $M$  do
11:   Calculate  $S(CRC_q, CRC_i)$  {using equation 3.1}
12:   if  $(S > T_B)$  then
13:     Add  $CR_i$  to list of similar CRs  $SCR$ 
14:   end if
15: end for
16: for each  $scr \in SCR$  do
17:   Add all  $f \in scr_f$  to  $I_{BOW}$ 
18: end for
19:  $I_F = I_{BOW} \cap I_{TM}$ 
20:  $ICR = SCR \cap SL$ 
21: Sort  $ICR$  according to similarity values w.r.t  $CR_q$ 
22: for each  $icr \in ICR$  do
23:   Add  $icr_e$  to  $I_E$ 
24:   Add  $icr_d$  to  $I_D$ 
25: end for
26: return  $I_F, I_E, I_D$ 
```

---

sorted according to similarity value and then corresponding effort and duration values are added to the final list of impacted effort and duration values,  $I_E$  and  $I_D$  respectively. Here both  $I_F$  and  $ICR$  are generated by taking the *intersection* of files and CR appearing respectively, from BOW and topic similarity. An additional variant of *union* was also attempted, but *intersection* yielded better results in both cases. Thus this was retained for subsequent analysis.

### 3.2.3 Bag of Words & File Coupling (BOW & CO)

The third technique for impact prediction take into consideration the Coupling (CO) of files. This technique only changes the *Prediction* process. CO has been used in some literature to support code impact prediction [32, 42, 45]. This technique generate coupling information, following the steps mentioned below. This information is then integrated with Bag of Words (BOW) textual similarity technique, defined in algorithm 1. This technique only analyzes file information in, thus it cannot be used for effort and duration prediction. The coupling information (matrix) is generated using commit history (of a specific period), retrieved from code repository. Here, each commit is treated as separate transactions. Consequently, the files which appear together in the same commit are taken to be related in terms of change and thus they change together. From commit history, a support matrix  $S_{q \times q}$  is generated, where any matrix entry  $S(a, b)$  is defined as:

$$S(a, b) = \mathcal{N}(F_a \cap F_b) \quad (3.3)$$

where  $\mathcal{N}(F_a \cap F_b)$  is a total number of times files  $F_a$  and  $F_b$  have appeared in the same commit.  $|q|$  is the total number of files in the commit history time interval. The following is an example of a support matrix, which show three files have changed. The matrix denote that  $F3$  has not changed with  $F1$ ,  $F3$  changed twice with  $F2$  and finally,  $F3$  changed seven times in total (including changing alone by itself (appearing as single file in any commit)).

$$\mathbf{S} = \begin{matrix} & \begin{matrix} F1 & F2 & F3 \end{matrix} \\ \begin{matrix} F1 \\ F2 \\ F3 \end{matrix} & \begin{pmatrix} 2 & 2 & 0 \\ 2 & 4 & 2 \\ 0 & 2 & 7 \end{pmatrix} \end{matrix}$$

To compensate for any bias [32, 42, 45], confidence matrix  $C_{q \times q}$  is generated (by normalizing support values with total counts of change), where entry  $C(a, b)$  is defined as:

$$C(a, b) = \frac{S(a, b)}{S(a, a)} \quad (3.4)$$

where  $S(a, a)$  is the total number of times that file  $a$  has changed.

For generating final list of impacted file, the following steps are augmented to Algorithm 1. For each file  $f \in si_f$  (step 7), confidence values are retrieved from confidence matrix  $C_{q,q}$ . If the confidence value for any file is greater than  $T_{CO}$ , then corresponding file is added to the final list of impacted files  $I_F$ . The procedure of BOW & CO appear in the paper [43].

### 3.2.4 Alignment with objectives

Preceding segments describe the design of proposed techniques for vector impact prediction of CR. The techniques primarily utilize CR repository data which have lower churn rate, compared to code data. They are simple, intuitive and easy to understand. They do not require any additional input from the users, reducing additional cost and over heads. The designs fulfill objectives RO2, related to design of the proposed system discussed in this chapter.

### 3.2.5 Evaluation measures

For studying the performance of the different techniques, the following evaluation measures are used. Recall, Precision and F-score are widely used measures in Information Retrieval studies [12], similar to this thesis. They are used for gauging effectiveness of file impact prediction accuracy. For a given dataset, these values are calculated for each query CR. The final value is the average of the values obtained from individual queries. The file impact prediction evaluation measure are also identical to the ones reported in [43].

- Precision  $P$  is the ratio of the number of correctly identified files to the total number of files (relevant and irrelevant) recommended i.e., retrieved files.

$$P(CR_q) = \frac{\#(\{relevant\ files\} \cap \{retrieved\ files\})}{\#\{retrieved\ files\}} \quad (3.5)$$

- Recall  $R$  is the ratio of the number of correctly identified files to the total number

of relevant files.

$$R(CR_q) = \frac{\#(\{\text{relevant files}\} \cap \{\text{retrieved files}\})}{\#\{\text{relevant files}\}} \quad (3.6)$$

- *F-score* combines the Precision and Recall, where the weights of each depends on the value of  $\beta$  (equation 3.7). Traditionally *F1* scores are mostly used (being the harmonic mean of Precision and Recall). However, some literature such as [61], [25] suggest that Recall is more important than Precision. Therefore, *F2*, *F5* and *F10* measures are also calculated, which emphasize more on Recall.

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{Precision} \cdot \text{Recall}}{\beta^2 \cdot \text{Precision} + \text{Recall}} \quad (3.7)$$

For assessing performance of effort and duration prediction, Magnitude of Relative Error (MRE) [88] is used, which is widely used for effort and duration prediction and utilized many of the related literature such as [27, 72, 94]. For a change request  $CR_q$ , MRE value is calculated as follows:

$$MRE(CR_q) = \frac{\text{PredictedValue}(CR_q) - \text{ActualValue}(CR_q)}{\text{ActualValue}(CR_q)} \quad (3.8)$$

where  $\text{PredictedValue}(CR_q)$  and  $\text{ActualValue}(CR_q)$  represent predicted and actual values (effort or duration) of the change request.  $\text{PredictedValue}(CR_q)$  of effort and duration is calculated from top  $N$  analogies, following the process discussed previously. An error bound  $E$  is defined, which define the tolerance threshold i.e., accepted difference between actual and predicted values. With respect to  $E$ , prediction accuracy is defined as below:

$$\text{Prediction}(CR_q) = \begin{cases} 1 & \text{if } MRE(CR_q) < E \\ 0 & \text{otherwise} \end{cases} \quad (3.9)$$

The final accuracy is calculated by testing over set of all available change requests. The final accuracy represents percentage of the test cases (query CR) where predicted value was within error bound  $E$ . It is formally defined as the following:

$$Accuracy(CR_M) = \frac{\sum_{cr \in CR_M} Prediction(cr)}{|CR_M|} \quad (3.10)$$

where  $CR_M$  represents the set of all available change requests.

### 3.3 Summary

This chapter outline the proposed approach in detail. Here three textual similarity based techniques for file prediction are presented. The models are based on Analogy Based Reasoning. The knowledge base is constructed using past CR textual data and information on the files they impacted. For a query, textual similarity is used to define analogy with previous solved instances to make predictions for the query. The three techniques all use the Bag of Words (BOW) representation. BOW utilize textual similarity to find similar cases. The first technique only utilizes textual similarity of BOW representation of change requests to make the prediction. The second technique combines textual similarity with topic similarity, with the aid of topics generated using Latent Dirichlet allocation (LDA). Finally, the third model combine file coupling information with textual similarity to make the prediction. Two of the techniques defined for file impact prediction are extended for effort and duration prediction. These models are BOW and the combination of BOW with LDA topics. The techniques presented, help to achieve objective RO2. This chapter concludes with definitions of the evaluation measures for measuring the performance of the proposed techniques in different experimentation.

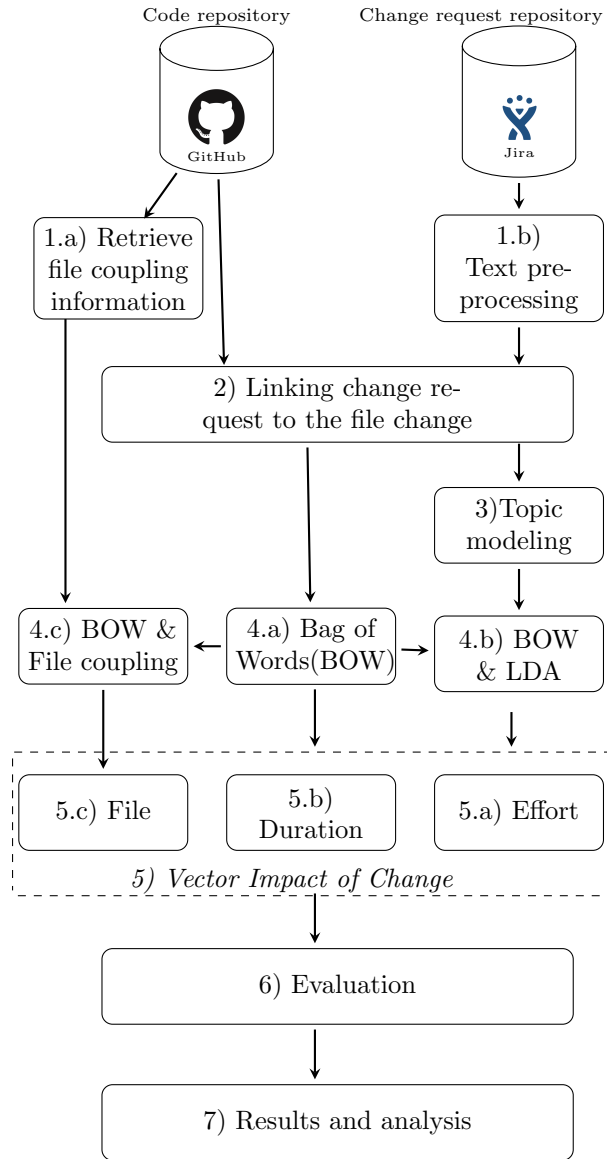


Figure 3.1: Process outline of this thesis (adapted from [43]).



## Chapter 4

### Case Studies

Case studies have evolved to be a powerful and popular empirical technique, being used by a plethora of researchers. A case study aims to understand and explore a particular phenomenon or to establish a theory [70]. Like other disciplines, case study has become popular in software engineering. It is used to comprehend, explain and exhibit potential of any new technique, method, tool, technology, process [86]. In this thesis, two case studies are performed. The first case study is undertaken for evaluating the performance of VCIA on two real-world industry projects. The second case study analyzes the performance of VCIA on two popular OSS projects. Case studies are designed towards achieving objective RO3. Table 4.1 shows the key attributes of each case study. Studies are conducted following guidelines for conducting case studies in software engineering, as demonstrated in [81]. The following sections discuss on each of the mentioned case studies. For each case study, the objective, context, result and threats to validity are mentioned.

#### 4.1 Case study 1: Evaluation of VCIA on real-world industry projects

Case studies like this one is *descriptive* in nature, and investigates a phenomenon in the real-life [86]. In this software engineering study, the case under review is *vector impact of change*. Case has been derived from two real-world software projects titled Mail and Dental. The projects belong to BrightSquid. According to the classification and guidelines presented in [82], this is a *multiple-case holistic* case study. This case study address objective RO3.1.

Some contents in the following sub sections (related to case study 1) come from the paper [43], which reports the application of VCIA in Brightsquid. In particular the data analyzed, defined research questions, process overview, evaluation measures and other related entities all come from the paper [43]. In most cases some modifications were made to the paper components before they

Table 4.1: List of case studies and corresponding Research Questions.

Case Study Number	Research Questions
Case Study 1: Evaluation of VCIA on two projects from Brightsquad	<p>-CS1.1: Among the three techniques (i) Bag of words (BOW) (ii) Bag of words and Latent Dirichlet Allocation (BOW &amp; LDA) and (iii) Bag of words and file coupling (BOW &amp; CO), which one works best better for predicting impacted files by a change request?</p> <p>-CS1.2: How the file impact prediction techniques perform in dependence to the type of change request?</p> <p>-CS1.3: How well can BOW and BOW &amp; LDA technique, as defined in CS1.1 be used for predicting effort and duration of implementing a change request?</p> <p>-CS1.4: What is the impact of the number of recommendations on files impacted, effort and duration prediction?</p>
Case Study 2: Evaluation of VCIA in OSS environment	<p>-CS2.1: How well can VCIA predict the vector impact of the change request in OSS projects?</p> <p>-CS2.2: How do the results obtained using industry project data compare with those obtained using OSS project data?</p>

were placed in this thesis. The results for file impact prediction in this case study are identical to the paper. However, for duration and effort prediction, the evaluation measures used in this thesis is different, thus the results in the thesis are different from the paper. In any case, the components from the original paper (with or without modifications) are cited with reference to the paper, as they share the same core essence. This includes tables and figures from the paper [43]. For most of the cases, the description in this thesis provide a more detailed account compared to the version in the paper.

#### 4.1.1 Context of the study

Brightsquad is a rapidly growing start-up, specializing in health informatics. They provide solutions for secure clinical communications among doctors, patients, and support staff. The core product is a secure collaboration platform called Secure-Mail. This platform allows efficient patient care management, results sharing, and quick allocation of expertise.

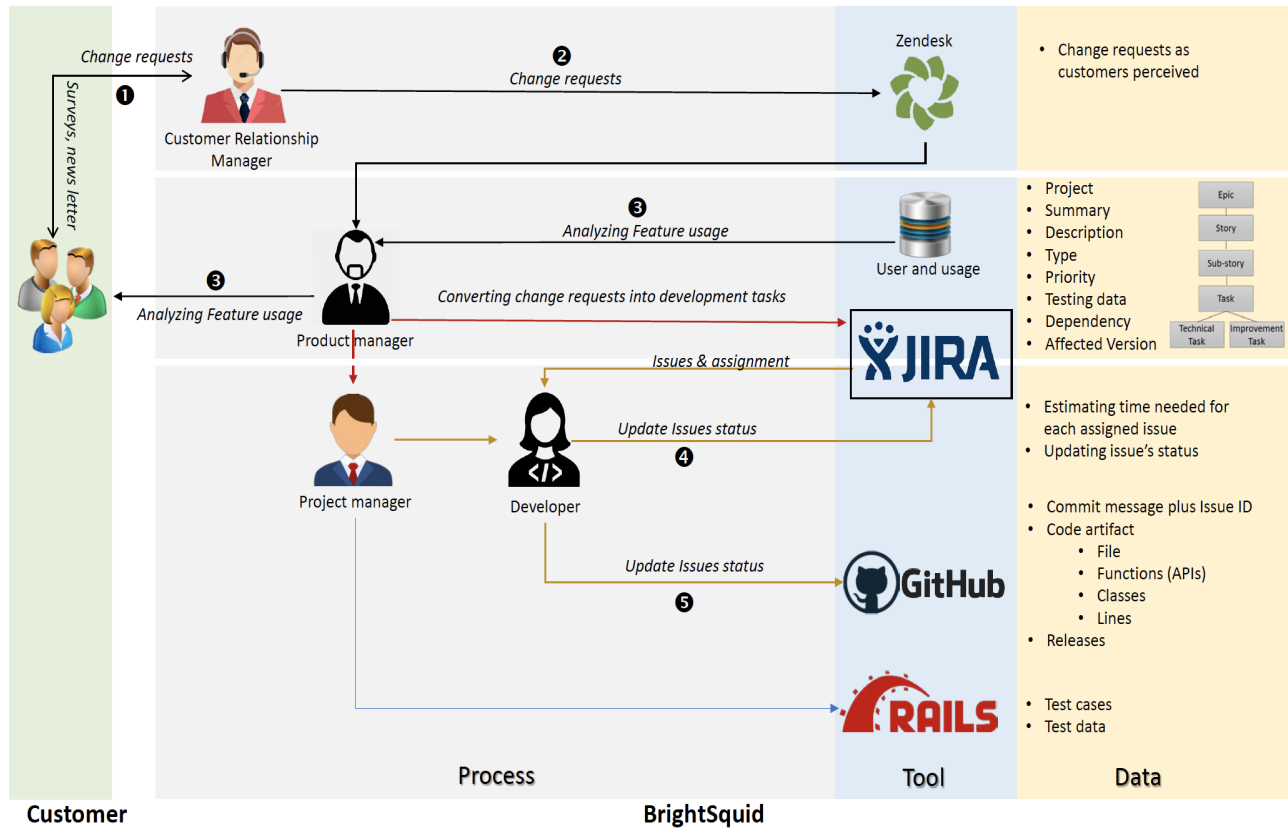


Figure 4.1: Process model at Brightsquad.

Figure 4.1 shows the process model followed by Brightsquad. Customer feedback, complaints and feature requests are collected through calls, surveys, newsletters etc. These are then analyzed to create change requests, which are stored in the Zendesk repository. In addition these requests, system usage is monitored and tracked. The CRs and usage are analyzed primarily by the Product Manager, who requests these requests and statistics into development tasks. The tasks are grouped into *Story* and *Sub-story*, which form the basis of different releases. All development tasks are stored in Jira issue management system. The development tasks gets forwarded to the development team, who are managed by the Project Manager. Figure 4.2 shows Brightsquad's change management process. Brightsquad follows a Scrum methodology for software development. The products are implemented and refined incrementally through multiple sprints. Duration of a sprint is usually two weeks. The core product is offered as web, desktop, and mobile application services.

Software functionality is implemented across thousands of files, resulting in complex interacting entities. Issues in the backlog are analyzed with respect to the files changes, effort and duration required. Issues are then subsequently assigned to developers. Developers work on implementing these tasks, making updates to Jira as the tasks progress and eventually completes. Depending on nature, each request might require changes to one or multiple files. Envisioning file changes, with predicting effort and duration, is mostly subjective to the experience of solving similar problems in the past. Implementation of the task requires changes to the code base of Brightsquid, which is maintained in GitHub version control system. It requires careful navigation and expertise of the complex interaction of the code and issue entities. Also, due to the size of the system, it is inefficient, even for experienced developers to locate the scope of these changes manually. The dashed box labeled Impact Analysis in Figure 3.1, is where a semi-automated solution would be most helpful towards change impact identification and management.

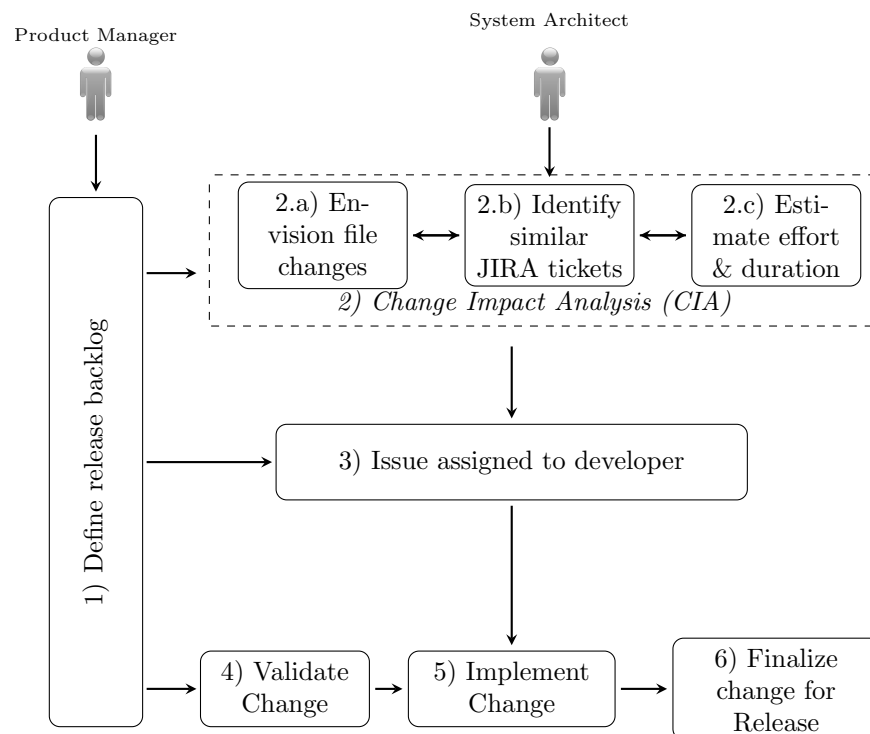


Figure 4.2: Change management process at Brightsquid (adapted from [43])

#### 4.1.2 Data collection

Data for the first case study originates from two Brightsquid projects i.e., Mail and Dental. Table 4.2 summarizes the data extracted. Mail is a more recent project, compared to Dental which is older in terms of creation date. The projects were analyzed between July, 2015 to May, 2017. A total of 1116 change requests were retrieved from Mail project between May 2016-2017. These change requests could be traced back to 2957 commits which impacted a total of 717 files. Similarly, 528 change requests were retrieved from the Dental project between July 2016 - 2017. These change requests from were traceable to 1118 commits which impacted a total of 1458 files. The data discussed and summarized in Table 4.2 is used to perform the experiments of case study 1. The same data is also used in [43].

#### 4.1.3 Initialization & parameter setting

Leave-One-Out Cross Validation (LOOCV) is used to evaluate performance of the model. In each iteration, one CR acts as the test case, while the remaining change requests are used for generating prediction (training). For running the model, corresponding file changes are captured from GitHub. Files which have changed only once in the analysis time span, are omitted from analysis. The assumption is if file changes only once in this duration, it is unlikely to change again. Based on various trials and error combinations, the value of  $T$  is set to 0.3 as it presented with the best trade-off. Similarly,  $T_{CO}$  is set at 0.4 for both Dental and Mail projects (see figure A.5). Finally, the error bound defined in equation 3.9 is set to  $E = 0.25$ . Setting error bound is based on discussion with industry partner and experimentation. The initialization and parameter settings are very similar to the setup used in [43].

Table 4.2: Time frame and number of change requests for vector impact prediction in industry projects [43].

Project Name	Time Frame	Number of change requests	Number of commits	Number of files
Mail	May 2016 - May 2017	1116	2957	717
Dental	July 2015 - July 2016	528	1118	1458

#### 4.1.4 Case study objective

This case study evaluates and analyzes the performance of VCIA with respect to prediction abilities in real world industry projects. Key factors and attributes affecting the performance are analyzed and presented. The discussion of this case study begins with the research questions (adapted from questions in [43]), which are defined below:

**CS1.1: Among the three techniques (i) Bag of words (BOW) (ii) Bag of words and Latent Dirichlet Allocation (BOW & LDA) and (iii) Bag of words and file coupling (BOW & CO), which one works better for predicting impacted files by a change request?**

**CS1.2: How the file impact prediction techniques perform in dependence to the type of change request?**

**CS1.3: How well can BOW and BOW & LDA technique, as defined in CS1.1 be used for predicting effort and duration of implementing a change request?**

**CS1.4: What is the impact of the number of recommendations on files impacted, effort and duration prediction?**

#### 4.1.5 Value of predicting vector impact of change

Existing impact analysis techniques look at code change independently and does not provide much insight into the effort or duration that might be required to implement a change request. Without the additional effort and duration information, feasibility analysis would be much more difficult because time and human resource usage cannot be properly measured. The following points describes the value of including duration and effort estimations into the change management decision making, which also appear in [43].

- *Cost/Benefit decisions:* Developers, managers, and stakeholders will be able to perform cost/benefit analysis. They will be able to judge what would be the perceived benefit of implementing a change request versus the cost and time required to achieve the change. This allows decision makers, particularly managers to gauge

the perceived benefits of cost and time considerations. Priority can be provided to features which would rank high on benefit and low on cost. Essential core-features can be better emphasized, which is crucial for new and small software enterprise, with focus on MVP.

- *Making commitments:* Having an estimation on the duration i.e., time for completing a change would make it easier to determine release and delivery schedule. Managers can have a better idea of how long it will take to complete change, and setup milestones and deliverable accordingly.
- *Allocation of staff and resources:* Having effort and duration predictions for change requests would make allocating scarce resources less tedious and error-prone. Various development activities can be efficiently planned within time and resource constraints [46].
- *Customer satisfaction:* For a given change request, compared to manual estimations, vector CIA will help to make more realistic release decisions. For future releases, this includes the setting of more realistic proposed feature-set and release dates. For ongoing maintenance activities, vector prediction will help to reduce problem fixing times, allowing for better conformance to service level agreement. All of these increase the likelihood of achieving customer satisfaction, making it easier for Brightsquid to maintain the market reputation and instill consumer confidence.
- *Forecasting Return on Investment:* It would be easier to estimate expenses, helping to forecast revenue streams. All of these would result in a more accurate analysis of financial growth and return on investment over time.

#### 4.1.6 Analysis of results

The subsection presents the results of the outlined analysis. The results are grouped and presented according to defined research questions in the following segment of this chapter. As mentioned already, results of file impact prediction are identical to those reported in [43].

**CS1.1: Among the three techniques (i) Bag of words (BOW) (ii) Bag of words and Latent Dirichlet Allocation (BOW & LDA) and (iii) Bag of words and file coupling (BOW & CO), which one works best better for predicting impacted files by a change request?**

Table 4.3 show the performance of the three techniques for file impact prediction. The performance of the techniques are reported with respect to Precision and Recall evaluation measures. The performance is also reported for different number of file recommendations  $F$ . Charts generated using the data from table is shown in figures 4.3 - 4.4. In the charts, suffix  $D$ - and  $M$ - correspond to results generated from Dental and Mail projects respectively.

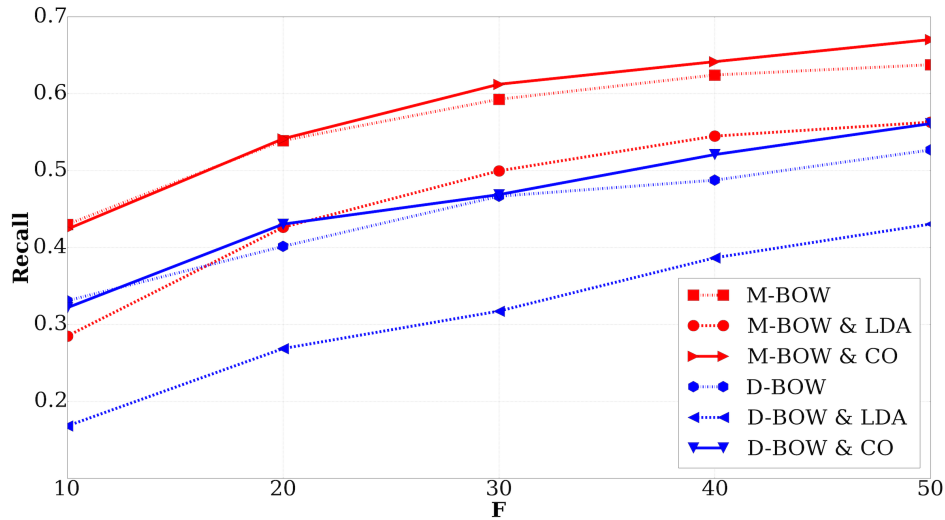


Figure 4.3: Recall of file impact prediction by applying the three proposed techniques on industry projects Mail (M) and Dental (D), in dependence to the number of file recommendations  $F$  [43].

In terms of Recall evaluation measure, the file coupling approach generates the best values. The trend is similar for both Mail and Dental projects. BOW & CO achieved the highest Recall of 0.67 and 0.56 for both Mail and Dental projects respectively. With respect to Recall, the second



Table 4.3: Accuracy of file impact prediction for the three proposed techniques applied to Mail and Dental projects and reported for different number of file recommendations  $F$  [43].

Technique	F	Mail					Dental						
		Precision	Recall	F1	F2	F5	F10	Precision	Recall	F1	F2	F5	F10
BOW	10	<b>0.11</b>	0.43	<b>0.17</b>	<b>0.27</b>	0.38	0.42	<b>0.12</b>	0.33	<b>0.17</b>	<b>0.24</b>	0.31	0.32
	20	0.07	0.54	0.12	0.23	0.43	0.51	0.08	0.40	0.13	0.22	0.35	0.39
	30	0.05	0.59	0.10	0.19	<b>0.43</b>	0.54	0.07	0.47	0.12	0.21	0.38	0.44
	40	0.04	0.62	0.08	0.17	0.41	0.55	0.06	0.49	0.10	0.19	0.38	0.45
	50	0.03	0.64	0.07	0.14	0.38	0.54	0.05	0.53	0.09	0.19	0.39	0.48
BOW & LDA	10	0.07	0.28	0.12	0.18	0.26	0.28	0.07	0.17	0.10	0.13	0.16	0.17
	20	0.06	0.43	0.10	0.18	0.34	0.40	0.06	0.27	0.09	0.15	0.24	0.26
	30	0.04	0.50	0.08	0.16	0.36	0.45	0.05	0.32	0.08	0.15	0.26	0.30
	40	0.04	0.54	0.07	0.14	0.36	0.48	0.04	0.39	0.08	0.15	0.30	0.36
	50	0.03	0.56	0.06	0.13	0.34	0.48	0.04	0.43	0.08	0.15	0.32	0.39
BOW & CO	10	<b>0.11</b>	0.42	<b>0.17</b>	<b>0.27</b>	0.38	0.41	0.11	0.32	<b>0.17</b>	0.23	0.30	0.32
	20	0.07	0.54	0.13	0.24	<b>0.43</b>	0.51	0.09	0.43	0.14	<b>0.24</b>	0.37	0.41
	30	0.06	0.61	0.10	0.21	0.44	0.56	0.07	0.47	0.12	0.22	0.39	0.44
	40	0.05	0.64	0.08	0.18	0.43	0.57	0.06	0.52	0.11	0.21	0.41	0.49
	50	0.04	<b>0.67</b>	0.07	0.16	0.41	<b>0.58</b>	0.06	<b>0.56</b>	0.10	0.20	<b>0.42</b>	<b>0.52</b>

best performing approach is basic BOW while the BOW & LDA trails in third.

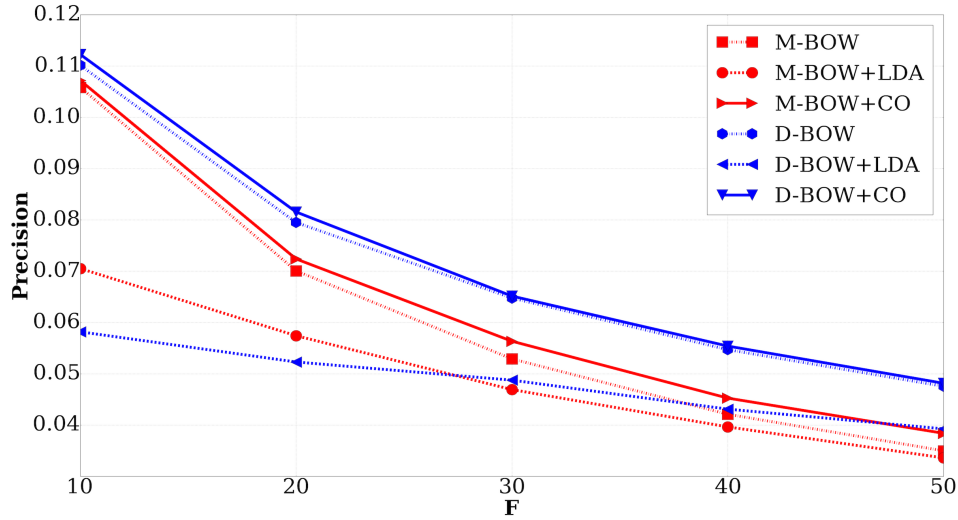


Figure 4.4: Precision of file impact prediction by applying the three proposed techniques on industry projects Mail (M) and Dental (D), in dependence to the number of file recommendations  $F$  [43].

Considering Precision evaluation measure, both BOW and BOW & CO perform similarly. The maximum value achieved is 0.12 for BOW only in Dental project. For the Mail project, BOW and BOW & CO both touch the maximum value of 0.11. BOW & LDA generates lowest Precision values for both projects.

As mentioned in chapter 3, F-score combines Recall and Precision values to form a combined evaluation score. For this case study, four separate F-scores are defined and calculated, i.e., F1, F2, F5, and F10. Considering F10 score, BOW & CO generate highest values, (Fig. A.1). Although BOW & CO generate lower precision values (compared to BOW and BOW & LDA), this technique is still the most valuable. This is because BOW & CO can retrieve the most number of impacted files compared to the other two techniques (higher Recall values compared to BOW and BOW & LDA). This is reflected in higher Recall values (and subsequently higher F5 and F10 values).

In general, the results indicate that file impact prediction can be improved if file coupling information is integrated with textual similarity. In the initial stage textual similarity retrieve important files. The coupling further augments initial file list with additional files, based on past co-change

pattern. Also, the coupling threshold restricts files with the lower degree of association for making into the final list. With respect to file impact prediction, a combination of BOW & LDA, the values are consistently lower. On average the values are 0.06 to 0.07 less than maximum observed values.

**Recall:** When file coupling was combined with simple textual similarity (BOW & CO), 0.67 (67%) of the impacted files could be correctly identified for  $F = 50$ . This is the best file impact prediction result from industry projects.

**CS1.2: How the file impact prediction techniques perform in dependence to the type of change request?**

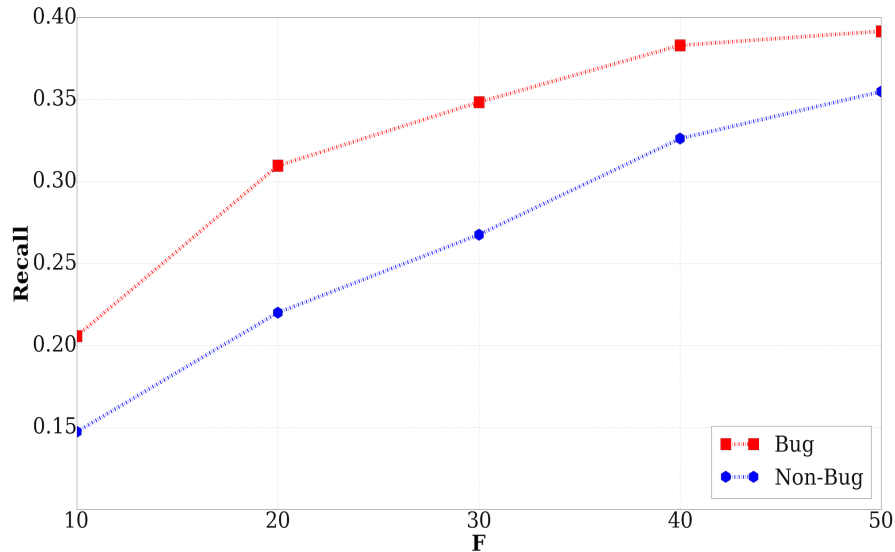


Figure 4.5: Recall of file impact prediction for Bug VS Non-Bug CR, achieved with BOW & CO applied on Dental project, in dependence to the number of file recommendations  $F$  [43].

Fig. A.2 and A.3 (in the appendix section) show the distribution of different CR types in Dental and Mail projects. From the pie-charts, it is clear that Bug type issues account for majority of change requests. To determine the effect of change request type on file impact results, BOW & CO file impact prediction approach was re-run with change requests separated into *Bug* (only Bug type issues) and *Non-Bug* (all issues whose type is not Bug). Figure 4.5 and 4.6 show the Recall values obtained when *Bug* and *Non-Bug* issues are separated. The values are higher for

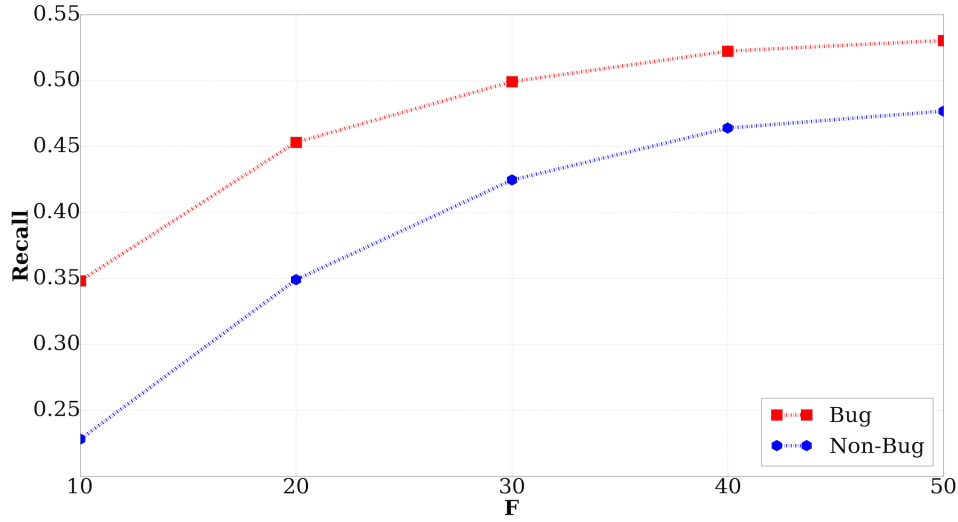


Figure 4.6: Recall of file impact prediction for Bug VS Non-Bug CR, achieved with BOW & CO applied on Mail project, in dependence to the number of file recommendations  $F$  [43].

both Mail and Dental. This is possible because Bug type issues are logged with better diligence, with changes being more systematically logged into the system.

**Change Request type:** *Bug* type change requests account for about 60% of all requests. File prediction results are better for *Bug* type issues.

**CS1.3: How well can BOW and BOW & LDA technique, as defined in CS1.1 be used for predicting effort and duration of implementing a change request?**

Table 4.4: Accuracy of effort and duration prediction for BOW and BOW & LDA techniques applied to Mail and Dental projects and reported for varying number of similar change request recommendations  $N$ .

N	Duration				Effort			
	Mail		Dental		Mail		Dental	
	BOW	BOW&TM	BOW	BOW&TM	BOW	BOW&TM	BOW	BOW&TM
1	0.19	0.17	0.24	0.22	0.27	0.30	0.29	0.36
3	0.26	0.24	0.30	0.28	0.40	<b>0.44</b>	0.42	<b>0.44</b>
5	<b>0.28</b>	0.26	<b>0.31</b>	0.29	0.41	0.44	0.41	0.42
7	0.28	0.26	0.31	0.29	0.41	0.44	0.40	0.43
9	0.28	0.26	0.30	0.29	0.41	0.44	0.40	0.43
11	0.28	0.26	0.31	0.28	0.41	0.44	0.42	0.42
13	0.27	0.25	0.30	0.28	0.41	0.44	0.41	0.42

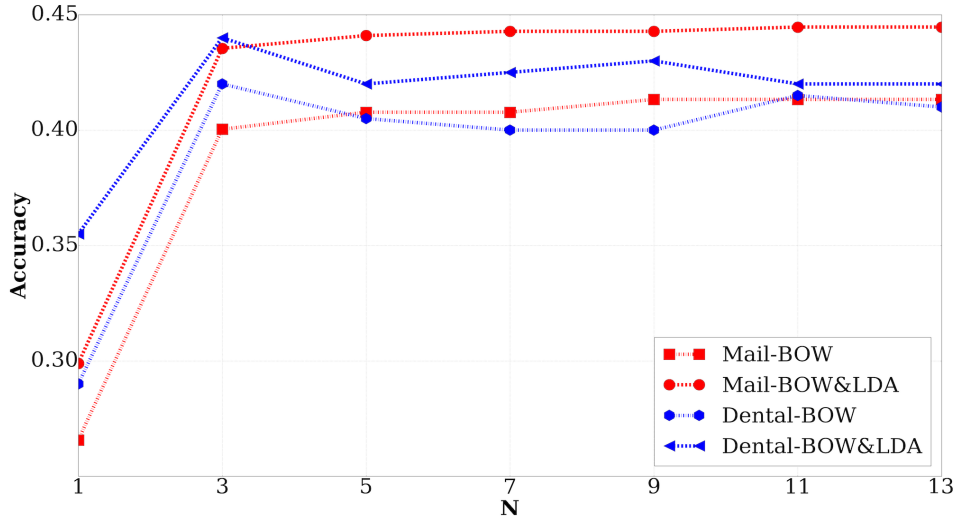


Figure 4.7: Effort prediction accuracy applying BOW and BOW & LDA on Mail & Dental projects, in dependence to the number of change request recommendations  $N$ .

Table 4.4 summarizes the accuracy values attained by the textual techniques for effort and duration prediction. Figure 4.7 and 4.8 show accuracy of effort and duration prediction respectively. Similar to file impact prediction, the value of  $N$  specify number of similar change requests (ranked according to textual similarity) considered for duration and effort predictions. With respect to duration, accuracy varies between 0.17 and 0.28 for Mail project. For Dental, duration accuracy ranges between 0.23 and 0.31. In general, better duration accuracy is observed for BOW approach. Similar analysis for effort estimation reveals that accuracy varies between 0.26 and 0.44 for Mail and, 0.29 and 0.44 for Dental projects. Considering the best-observed results, BOW & LDA outperformed BOW approach, with respect to effort prediction.

**Effort & duration prediction:** With respect to industry project data, BOW & LDA works best for effort prediction, while BOW works best for duration prediction.

**CS1.4: What is the impact of the number of recommendations on files impacted, effort and duration prediction?**

*File impact prediction:* In this study,  $F$  determines the size of the recommendation list. For

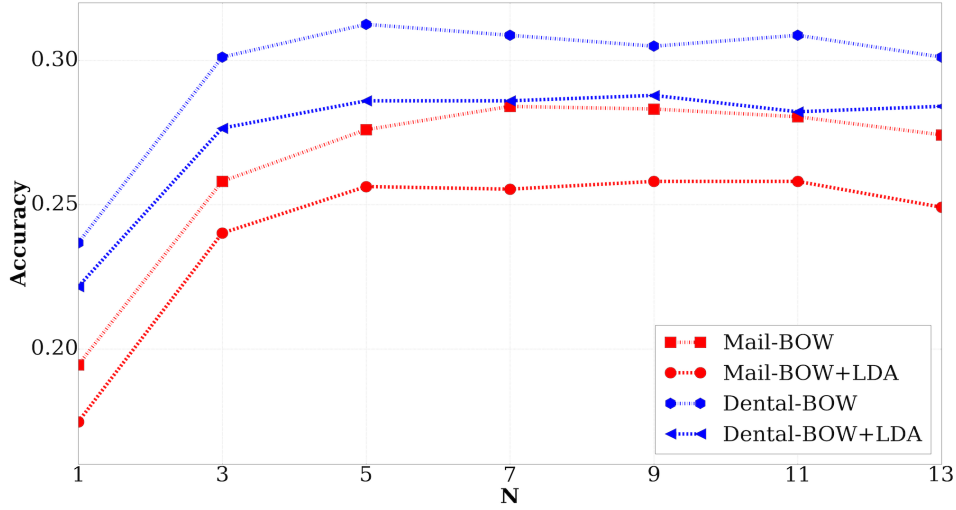


Figure 4.8: Duration prediction accuracy applying BOW and BOW & LDA on Mail & Dental projects, in dependence to the number of change request recommendations  $N$ .

example, if  $F = 8$ , the evaluation metric are calculated considering only first eight files coming from similar change requests. Increasing  $F$  positively impacts all evaluation measures except Precision. This is because, higher  $F$  results in more number of files in the recommendation list. This in turn causes the size of observed truth to go up. The trend is true for both Mail and Dental projects as well as across all the approaches presented. Conversely, growing  $F$  values cause the Recall to increase. The fact is visible for all higher F-score values. Similar trend is reflected in both Mail and Dental projects. Because increasing  $F$  leads to more results being considered, this increases the probability of file impact match. For all approach and project combinations the trend is clear i.e., increasing  $N$  increases number of impacted files observed. For increasing values of  $F$ , the inverse relationship between Precision and Recall is evident here and trade-off needs to be made depending on the scenario. For different experiments  $F$  has been varied between 10 and 50. For values between  $F = 30 - 40$  the highest F5 and F10 values are observed. The Recall value is also high in this range of  $F$ . This indicates that files can be relatively well predicted by analyzing 30-35 file recommendations coming from top change requests.

**Number of recommendations:** For file impact prediction, increasing the number of file recommendations  $F$  has positive impact on Recall and F-score, while the impact on Precision is negative. In terms of effort and duration prediction, increasing the number of similar change request recommendations  $N$  has positive impact on the prediction accuracy.

*Effort & duration:* In this case,  $N$  defines the number of change requests to be used for effort and duration prediction. With increasing values of  $N$  between 1 and 3, there is a sharp upward shift in accuracy values for effort and duration. After  $N = 3$ , the values flatten down, with slight fluctuations. The experiment was repeated with  $N$  between 1 and 13. Analyzing the accuracy values, it is clear that although experiments have been done up to  $N = 13$ , reasonable results can be achieved by examining top 9 similar change requests. Based on this threshold, an average of 0.40 accuracy is achieved.

**Effort + duration prediction:** Considering only *Top 3 most similar change requests*, effort and duration can be predicted with 0.44 and 0.30 accuracy on Dental project (Table 4.4).

#### 4.1.7 Improvement for Brightsquid

Brightsquid is a small company, operating with seven technical employees for system development and maintenance. At the time of project initiation, Brightsquid's focus was primarily on pushing out software releases. There was little resource available in terms of man hours for monitoring processes and subsequent improvements. Brightsquid then decided to engage in a collaboration project, in a bid to improve their performance. The primary objective was to enhance return on investment, increase customers satisfaction and improve financial forecasting. An essential road-block towards attaining this aim was the change management process. The entire change impact assessment was done manually.

The author of this thesis was working as part of collaboration team with Brightsquid. The author was primarily tasked with CR analysis. In general, the idea was to study CR for identifying key

trends and patterns. Eventually, the study shifted towards change impact analysis because CR is the initiating point of change. Through regular bi-weekly meetings and discussions, it was found that Brightsquad was more interested towards identifying similar Jira tickets or CR from their repositories. They wanted to use this knowledge for sprint estimation and planning. A textual similarity based impact prediction technique was initially formulated for only file impact prediction. Subsequently, through multiple meetings and discussion sessions, the model was extended to include effort and duration prediction. Incrementally, the current VCIA approach has been designed and implemented. An online tool has been developed based on VCIA. Using VCIA, Brightsquad can achieve the following, that was previously not possible with the manual approach:

- Cost & duration of CR: Based on a change request, Brightsquad can identify similar CR with the associated cost and duration from the repository. The cost here is measured in terms of man-hours and duration in terms of days. Previously, Product and Project managers would manually scour the CR repository to identify similar instances. With the development of VCIA tool, this is done automatically. The tool also generates predicted values of cost and duration for the incoming CR.
- Predicting file change: Using VCIA developers and Project manager can predict the files, where changes would need to be made to satisfy the incoming CR. Previously identification of file impact was based on developers experience and knowledge of the system. VCIA automates this process and reduces the cognitive load by analyzing a large number of interacting entities.

Due to lack of data availability, it is difficult to measure if the return on investment, customer satisfaction or improvement in financial forecasting. Also, measurement of a qualitative attribute such as customer satisfaction is difficult and subjective. Therefore, it is challenging to show improvements concerning these attributes. However, Brightsquad does consider these results and VCIA tool to be valuable. The benefits and lessons learned from applying VCIA at Brightsquad is



reported in [43], where members of Brightquid are co-authors. In addition, Brightquid has allocated domain, space and access to their data for building the tool and performing various kinds of analysis. These indicate their interest and positive attitude towards VCIA in general. Compared to previously having to just rely on experience and gut feeling, there is now a systematic method which provides useful prediction and analytics result to complement expert knowledge and aid in decision making. Brightquid is additionally happy because these recommendations come with minimal additional involvement from their end. Currently, VCIA tool is being used to complement expert knowledge and decision making for estimating and planning for sprints. The improvements and benefits are also reported in [43].

## 4.2 Threats to validity

This section discusses the threats to validity related to case study 1. Based on the classification of threats to validity presented in [96], four threats to validity are identified for this study i.e., construct validity, conclusion validity, internal validity and external validity. The following subsections discuss each of them.

### 4.2.1 Construct validity

The key threat to construct validity is the reliance of the proposed techniques on *CR ID* for linking CR to file commits. This linking is used for generating file impact prediction. This linking process involves searching for *CR ID* in the commit messages. If any *CR ID* is found, then the files in the commits are linked to corresponding CR, whose ID has been identified. However, the presence of CR ID in the commit message does not always guarantee a relation. Developers undertake system improvement such as security upgrades or performance optimization. In such cases, few CR are logged into the issue management. Compared to ordinary CR, the number of files impacted (changed) against these improvement CR are very large. If these commits are included in the analysis, then file impact prediction will be incorrect, as the actual link between CR and files do

not exist. To overcome this situation, very large commits including *merge* and *initial* commits were manually analyzed to identify above mentioned scenario and subsequently discarded from the analysis.

Another threat related to the first one is that not not all file change can be linked back to the commits, because the *CR ID* is not mentioned in commit message. Analysis has revealed that only about one-third of CR are traceable for Mail and Dental industry projects. Having higher traceability positively impact the performance, because more training examples would be available for prediction. In order to ensure there were enough traceable CR, time period after *May 2016* was chosen, this is because after the mentioned date Brightsquad encouraged and monitored the number of CR which were traceable to commits.

Another important threat to construct validity is the assumption that, if two or more files appear together in the same commit, then the files changes are related, and the corresponding files should be considered as “coupled”. This is also true for all related literature which utilize file file coupling. Files may appear to be coupled together for many scenario such as the one mentioned in the previous threat. In order to ensure that files appearing together are indeed coupled and not just appearing together due to other circumstance,  $T_{CO}$  is set at 0.40. This implies, that a file A will be considered as coupled to another file B, if B changed at least 40% of the times that A changed. Setting of this threshold helps to ensure that files appearing together due to other reason except actual *coupling*, may be filtered out from the analysis.

#### 4.2.2 Conclusion validity

Conclusion validity refers to correctness of the conclusions that are derived from the results. The most important element of conclusion validity are the selection of the evaluation measures, because they define the effectiveness of the results obtained. To mitigate any risk related to choice of the evaluation measures, popular and extensively used evaluation measures have been considered. The evaluation measures Precision, Recall. F-score and MRE accuracy have been used by related literature. In addition, the choice of evaluation measures (including choice of  $E$  in MRE accuracy)

have all been reviewed and accepted by industry experts at Brightsquad.

Another important criteria is the selection of various parameters, which has a strong influence on construct validity. This is because they impact the execution process of VCIA directly. To minimize risks related to incorrect selection of parameter values, related literature have been studied and their suggestion of key attribute values were considered. In addition, experiments were repeated with different values to ensure that best results could be achieved.

#### 4.2.3 Internal validity

Internal validity refers to correctness of the results. VCIA has been applied on two real world industry projects which are actively maintained and utilized. In addition the result generation and evaluation process is clearly presented for this case study. Thus, the threats to internal validity have been minimized.

#### 4.2.4 External validity

The results presented previously in this chapter are based on this industrial case study only i.e., Brightsquad project data only. It is therefore difficult to ensure external validity. Evaluation and analysis of VCIA on additional industry data is required for ensuring external validity of results. However, the results indicate that VCIA is able to predict the vector of change to a good extent from industry data. The VCIA tool can aid the process of change decision making and it is being utilized as part of an analytics dashboard at Brightsquad.

### 4.3 Case study 2: Evaluating VCIA in OSS

#### 4.3.1 Case study objective

The objective of this second case study analyzes the performance of VCIA in OSS projects. Real-world industry projects are different to open-source, with respect to change requests and files impacted. This case study has been designed towards achieving objective RO3.2. Initialization and

parameter settings are same as case study 1. Like case study 1, the following research questions have been formulated:

**CS2.1: How well can VCIA predict the vector impact of change request in OSS projects?**

**CS2.2: How do the results obtained using industry project data compare with those obtained using OSS project data?**

#### 4.3.2 Context of the study

Data for this case study comes from two very popular OSS projects, freely available online. The selected OSS projects are briefly summarized as below:

- Apache Lucene: Apache Lucene <sup>1</sup> is a rich text-based search engine library developed by Apache Foundation. It is widely being used across diverse platforms. Lucene is very popular with object-oriented programmers since the entire library is written in Java. Issues are shared openly via an unrestricted Jira interface. Similarly code base shared through a public GitHub repository. We refer to this project as Apache
- DSpace: Application for building open repositories, supported and developed by the Duraspace <sup>2</sup> initiative. DSpace has a large community base of users involved with open source development practice. Similar to the first project, change request and code repository are publicly shared via open Jira and GitHub interfaces.

#### 4.3.3 Data collection

Data for this second case study originates from two open source projects i.e., Apache Lucene and DSpace. Table 4.5 summarizes the data extracted. A total of 344 change requests were retrieved from Apache Lucene project between August 2016 - August 2017. These change requests could

---

<sup>1</sup><https://lucene.apache.org/core/>

<sup>2</sup><http://www.dspace.org/>

Table 4.5: Time frame and number of change requests for vector impact prediction in OSS projects.

Project Name	Time Frame	Number of change requests	Number of commits	Number of files
Apache Lucene	Aug 2016 - Aug 2017	344	2016	609
DSpace	Aug 2016 - Aug 2017	398	455	1027

be traced back to 2016 commits which impacted a total of 609 files. Similarity, 398 change requests were retrieved from DSpace project between the same time frame. These change requests from were traceable to 455 commits which impacted a total of 1027 files. The data discussed and summarized in Table 4.2 is used to perform the experiments of case study 2.

#### 4.3.4 Analysis of results

The subsection presents the results of the outlined analysis. Results are grouped and presented according to defined research question, in the following segment of this chapter.

##### CS2.1: How well can VCIA predict the vector impact of change request in OSS projects?

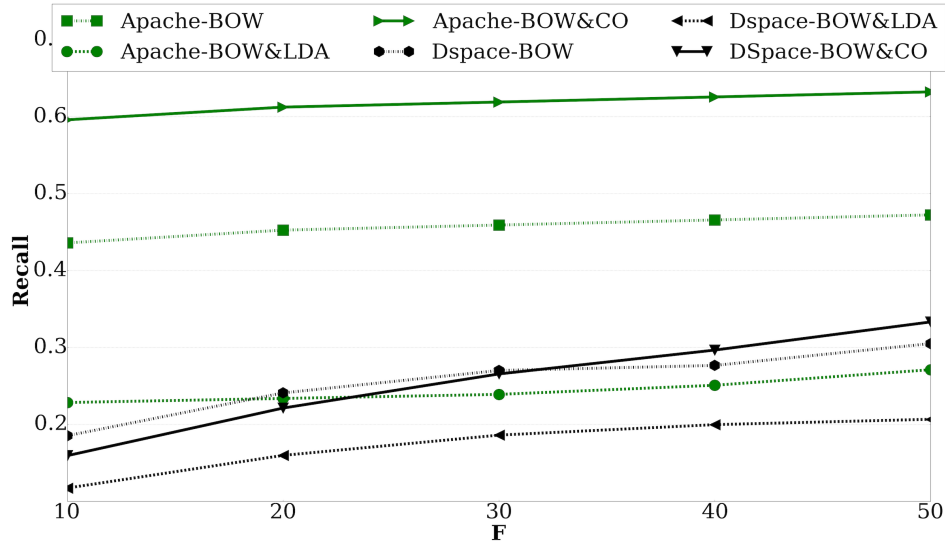


Figure 4.9: Recall of file impact prediction by applying the three proposed techniques on Apache and DSpace, in dependence to the number of file recommendations  $F$ .

*File impact prediction:* In this section the performance of VCIA on OSS projects are evaluated.

Table 4.6 shows file impact performance of the three techniques. Figures 4.9, 4.10 and A.4 have

Table 4.6: Accuracy of file impact prediction for the three proposed techniques applied to Apache and DSpace projects and reported for varying number of file recommendations  $F$ .

Technique	F	Apache-Lucene					DSpace						
		Precision	Recall	F1	F2	F5	F10	Precision	Recall	F1	F2	F5	F10
BOW	10	<b>0.26</b>	0.44	<b>0.33</b>	0.38	0.42	0.43	<b>0.11</b>	0.19	<b>0.13</b>	0.16	0.18	0.18
	20	0.23	0.45	0.31	0.38	0.44	0.45	0.09	0.24	0.13	<b>0.18</b>	0.23	0.24
	30	0.22	0.46	0.30	0.38	0.44	0.45	0.07	0.27	0.11	0.17	0.24	0.26
	40	0.22	0.47	0.29	0.38	0.45	0.46	0.06	0.28	0.10	0.16	0.24	0.27
	50	0.21	0.47	0.29	0.38	0.45	0.47	0.06	0.31	0.09	0.16	0.26	0.29
BOW & TM	10	0.21	0.23	0.22	0.23	0.23	0.23	0.07	0.12	0.08	0.10	0.11	0.12
	20	0.19	0.23	0.21	0.22	0.23	0.23	0.06	0.16	0.09	0.12	0.15	0.16
	30	0.18	0.24	0.20	0.22	0.24	0.24	0.05	0.19	0.08	0.12	0.17	0.18
	40	0.17	0.25	0.21	0.23	0.25	0.25	0.04	0.20	0.07	0.12	0.18	0.19
	50	0.17	0.27	0.21	0.24	0.27	0.27	0.04	0.21	0.06	0.11	0.18	0.20
BOW & CO	10	0.23	0.60	<b>0.33</b>	<b>0.45</b>	0.56	0.59	0.09	0.16	0.11	0.14	0.15	0.16
	20	0.21	0.61	0.32	0.45	0.57	0.60	0.08	0.22	0.12	0.16	0.21	0.22
	30	0.21	0.62	0.31	0.44	0.58	0.61	0.07	0.27	0.11	0.17	0.24	0.26
	40	0.21	0.63	0.31	0.45	0.58	0.61	0.07	0.30	0.11	<b>0.18</b>	0.26	0.29
	50	0.21	<b>0.63</b>	0.31	0.45	<b>0.59</b>	<b>0.62</b>	0.07	<b>0.33</b>	0.11	0.18	<b>0.29</b>	<b>0.32</b>

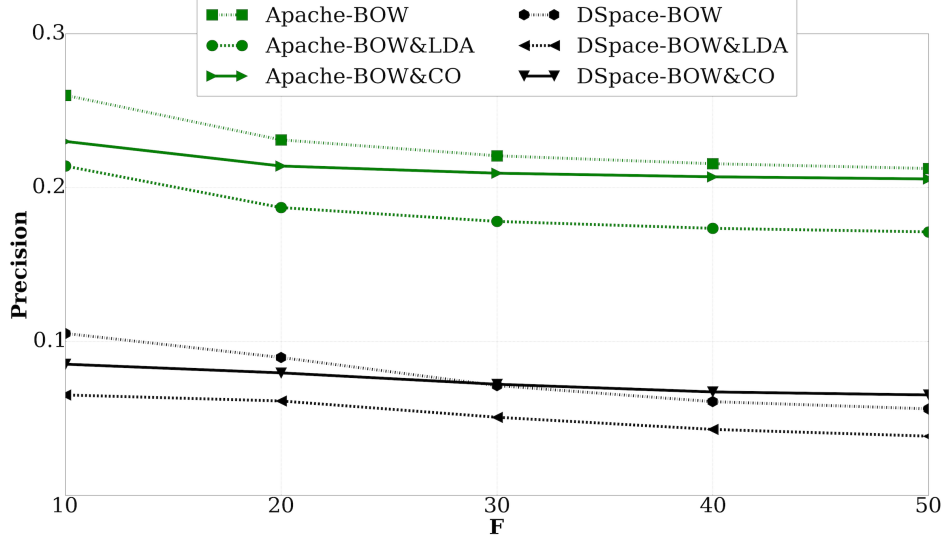


Figure 4.10: Precision of file impact prediction by applying the three proposed techniques on Apache and DSpace, in dependence to the number of file recommendations  $F$ .

been generated using values from the table, they show Recall, Precision and F10 values respectively. In Apache project, BOW & CO consistently performs better in terms of Recall, attaining highest value of 0.63. Recall values for BOW remains constantly in second best position, with maximum attained value of 0.47. BOW & LDA comparatively attains lowest Recall values, with maximum attained value of 0.27.

In DSpace project, the trend is almost very similar. Before  $N = 30$ , BOW performs a little better compared to BOW & CO, but for  $N$  greater than 30, BOW & CO again performs better. Similar to the previous OSS project, BOW & LDA generate lowest Recall values. The highest Recall values attained by BOW & CO, BOW and BOW & LDA are 0.33, 0.31 and 0.21 respectively in DSpace.

The values for F10 scores also show similar trends. Figure A.4 (in Appendix) shows the F10 scores achieved by each of the techniques. For Apache project BOW & CO consistently performs best, followed by BOW and BOW & LDA respectively, with maximum attained F10 scores 0.62, 0.47 and 0.27 respectively. For DSpace, BOW performs marginally better for  $F$  less than 30, otherwise BOW & CO performs better. Overall the maximum values attained are 0.32, 0.29 and 0.20 respectively for BOW & CO, BOW and BOW & LDA respectively.

In terms of Precision, BOW performs a little better compared to BOW & CO, while BOW & LDA attains lowest values. This trend is true for both Apache and DSpace projects. BOW attains maximum values of 0.26 and 0.11 in Apache and DSpace respectively. BOW & CO attains maximum values of 0.23 and 0.09 in Apache and DSpace respectively. Lastly, BOW & LDA attains maximum values of 0.21 and 0.07 in Apache and DSpace respectively.

Similar to industry projects, generally increasing  $F$  has positive impact on the Recall and F-score. But has negative impact on Precision. Continuing the trend of industry projects, BOW & CO continue to perform better than the other two techniques and produce best values with respect to Recall and F-score. Like the industry project, although the performance is marginally less compared to BOW, overall BOW & CO is better, because it can retrieve more the impacted files than the other two techniques.

**File impact prediction in OSS projects:** BOW & CO retrieved most impacted files in OSS projects. The maximum attained Recall is 0.63 for Apache project at  $F = 50$ .

Table 4.7: Accuracy of effort and duration prediction for BOW and BOW & LDA techniques applied to Apache and DSpace projects and reported for varying number of change request recommendations  $N$ .

N	Duration				Effort			
	Apache		DSpace		Apache		DSpace	
	BOW	BOW&TM	BOW	BOW&TM	BOW	BOW&TM	BOW	BOW&TM
1	0.26	0.25	0.12	0.12	0.26	0.34	0.22	0.24
3	0.32	0.28	<b>0.22</b>	0.16	0.35	0.42	0.34	0.34
5	<b>0.33</b>	0.30	0.20	0.16	0.36	<b>0.43</b>	0.35	0.36
7	0.33	0.30	0.20	0.16	0.36	0.43	0.35	<b>0.37</b>
9	0.33	0.30	0.20	0.16	0.37	0.43	0.34	0.36
11	0.33	0.30	0.19	0.16	0.37	0.43	0.35	0.36
13	0.33	0.30	0.19	0.16	0.36	0.43	0.34	0.36

*Effort & duration prediction:* Figure 4.16 shows the performance of BOW and BOW & LDA for effort prediction in OSS projects. BOW & LDA technique consistently performs better in terms of effort prediction accuracy. Generally the performance improves with increasing  $N$ . The highest accuracy achieved by BOW & LDA is 0.43 and 0.37 for Apache and DSpace respectively. BOW



achieves highest accuracy of 0.36 and 0.34 respectively in Apache and DSpace respectively.

Figure 4.11 shows the performance of BOW and BOW & LDA for duration prediction in OSS projects. BOW technique consistently performs better in terms of duration prediction accuracy. Similar to effort prediction, accuracy improves with increasing  $N$ . The highest accuracy achieved by BOW is 0.33 and 0.22 for Apache and DSpace respectively. BOW & LDA achieves highest accuracy of 0.30 and 0.16 respectively in Apache and DSpace respectively.

Figure 4.16 shows the performance of BOW and BOW & LDA for effort prediction in OSS projects. BOW & LDA technique consistently performs better in terms of effort prediction accuracy. Similar to duration prediction, accuracy improves with increasing  $N$ . The highest accuracy achieved by BOW is 0.37 and 0.35 for Apache and DSpace respectively. BOW & LDA achieves highest accuracy of 0.43 and 0.37 respectively in Apache and DSpace respectively.

**Effort & duration prediction in OSS projects** BOW & LDA attains best effort prediction accuracy of 0.43 and BOW attains best duration prediction accuracy of 0.33, both for Apache OSS projects.

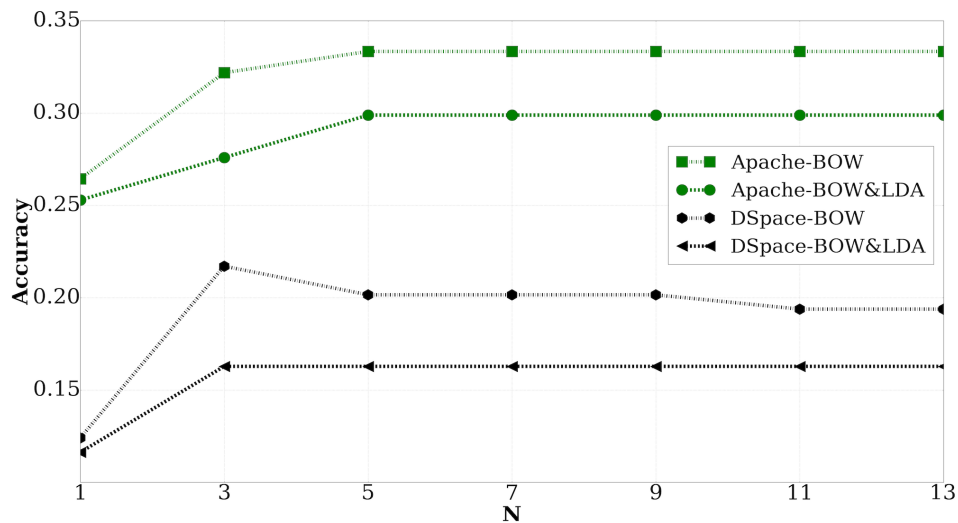


Figure 4.11: Duration prediction accuracy applying BOW and BOW & LDA on Apache & DSpace projects, in dependence to the number of change request recommendation  $N$ .

**CS2.2: How do the results obtained using industry project data compare with those ob-**

### tained using OSS project data?

In this sections, results obtained from industry and OSS projects are compared. In the fist segment, file impact prediction results are compared. The next segment compares effort and duration accuracy outcomes.

*File impact prediction:* Table 4.3 and 4.6 displays the results of file impact prediction by applying the three proposed techniques on industry and OSS projects respectively. As mentioned in the previous sections, BOW & CO identified most impacted files in both industry and OSS data experiments. Therefore in this segment, comparison across all projects is done with respect to BOW & CO. Figure 4.12 takes a closer look at Recall from BOW & CO across all projects i.e., OSS and real-world data. From the results, highest Recall of 0.67 is observed for Mail project. However the performance for Apache is more stable, with maximum value of 0.63 reached for this dataset.

**File impact prediction across all projects:** BOW & CO consistently retrieved most impacted files across all projects, with a combined maximum of 0.67 at  $F = 50$ , for Mail industry project.

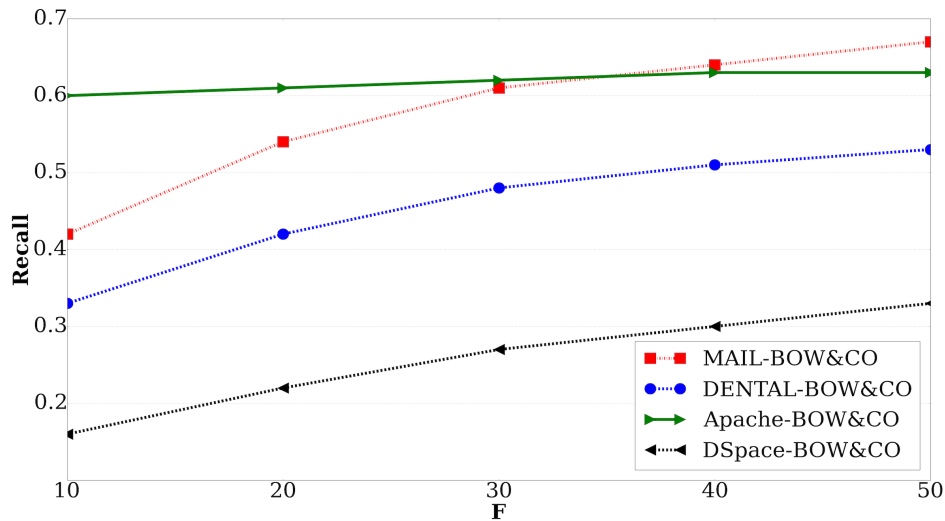


Figure 4.12: Recall of BOW & CO file impact prediction applied on OSS and industry projects, in dependence to the number of file recommendations  $F$ .

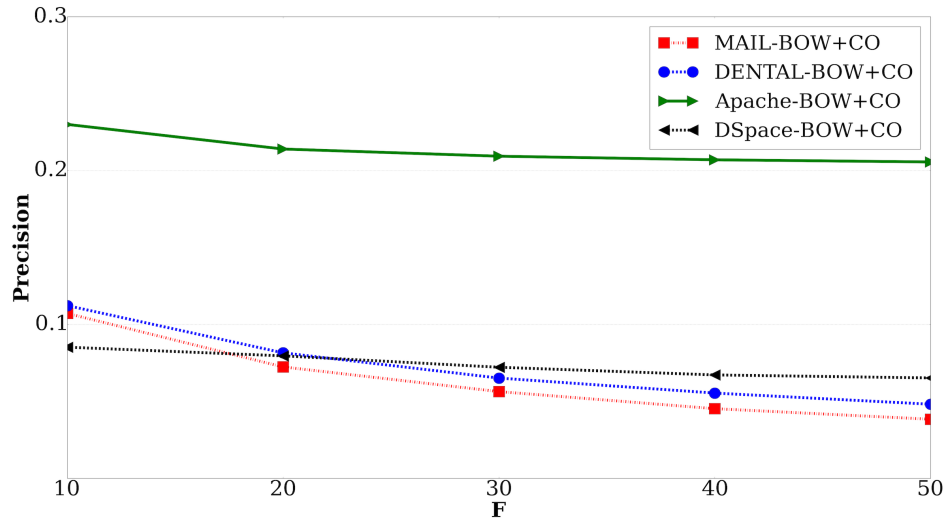


Figure 4.13: Precision of BOW & CO file impact prediction applied on OSS and industry projects, in dependence to the number of file recommendations  $F$ .

Comparing results of Precision, BOW & LDA generated the lowest values, while BOW generated slightly better results than BOW & CO. As BOW & CO returned highest Recall values (returned most impacted files), Precision performance comparison is also done with BOW & CO. The highest Precision value of 0.23 was attained on Apache project. For Mail & Dental projects, maximum Precision was 0.11 and finally for DSpace maximum Precision was 0.09. Figure 4.13 shows Precision attained by BOW & CO across all projects.

The performance comparison of F10 score was similar to Recall. Here again BOW & CO was chosen as the technique for comparison, across all projects. Maximum F10 score 0.62 was attained for Apache project. The other maximum attained F10 scores were 0.58, 0.48 and 0.32 for Mail, Dental and DSpace projects respectively.

*Effort & duration prediction:* Table 4.4 and 4.7 show the effort and duration prediction accuracy attained by BOW and BOW & LDA, for industry and OSS projects respectively. Mentioned in previous segments, BOW & LDA generated best effort prediction values. Figure 4.15 shows the performance of this technique across all projects. The maximum prediction accuracy of 0.44 was achieved for Mail and Dental projects. Compared to other projects, higher and stable accuracy

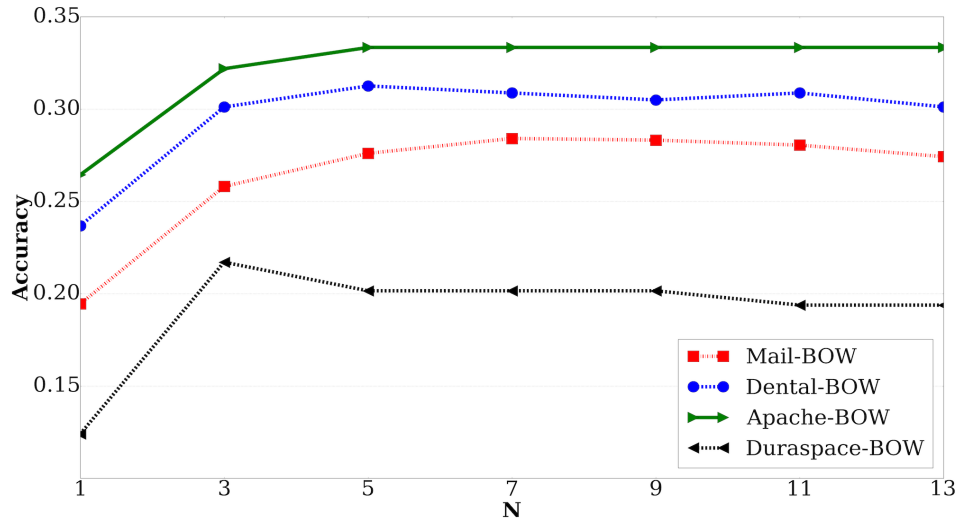


Figure 4.14: Accuracy of duration prediction by applying BOW on OSS and industry projects, in dependence to the number of change request recommendations  $N$ .

values are observed for Mail project. The other best values are 0.43 and 0.36 for Apache, DSpace respectively.

**Effort prediction across all projects:** Best combined effort prediction accuracy of 0.44 is achieved with BOW & LDA at  $N = 3$ , for Mail and Dental industry projects.

BOW generated best duration prediction accuracy values. Figure 4.14 shows the performance of this technique across as projects. The maximum prediction accuracy of 0.33 was achieved for Apache project. Compared to other projects, higher and stable duration accuracy values are found for Apache as well. The other best values are 0.28, 0.31 and 0.19, attained for Mail, Dental and DSpace projects respectively.

**Duration prediction across all projects:** Best combined duration prediction accuracy of 0.33 is achieved with BOW at  $N = 5$ , for Apache OSS project.

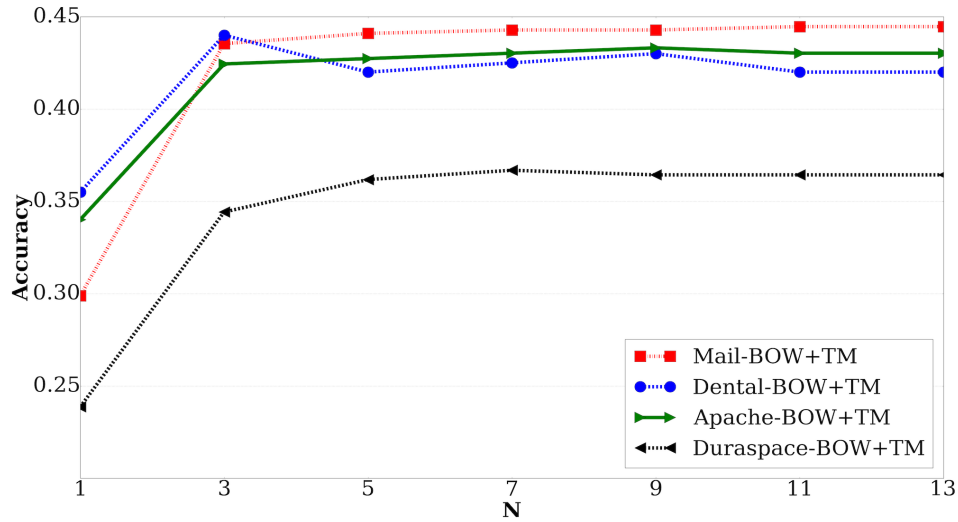


Figure 4.15: Accuracy of effort prediction by applying BOW & LDA on OSS and industry projects, in dependence to the number of change request recommendations  $N$ .

#### 4.4 Threats to validity

This section discusses the threats to validity, related to case study 2. Generally, the same construct, conclusion and internal threats mentioned in the first case study, apply here as well. Since the VCIA has only been applied and tested on two OSS projects, this represents a threat to external validity. However, to mitigate the threat the projects were chosen based on careful analysis and study. Both the OSS projects are active, well maintained and have large number of contributors. Moreover, they have been used in other literature to study various software engineering concepts, including the ones covered by this study.

Another important threat of this case study refers to the availability of CR effort estimation data. For many of the CR in OSS projects, effort estimation data is missing. For OSS data, it was very difficult to find any CR with effort estimates. This poses additional threat to conclusion and internal validity. To overcome this threat, CR from other projects, but within the same repository were retrieved for effort prediction.

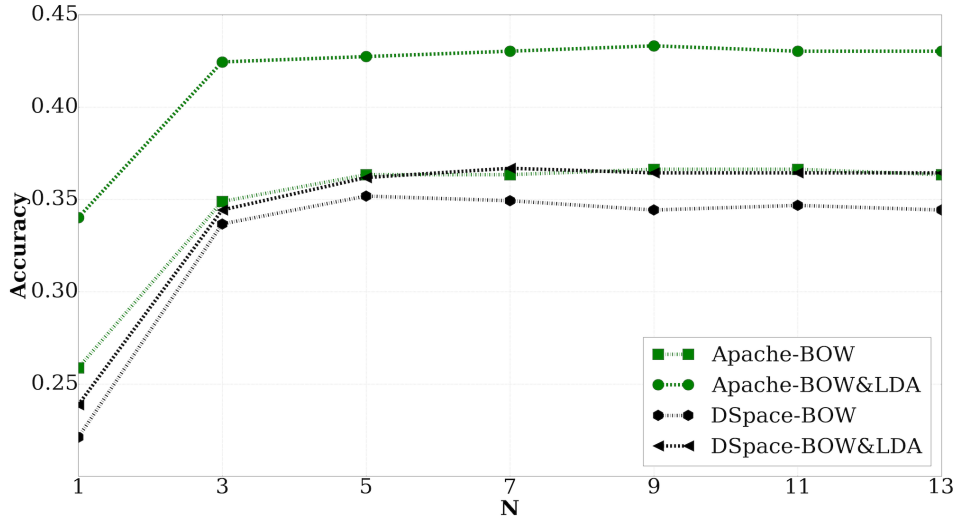


Figure 4.16: Effort prediction accuracy applying BOW and BOW & LDA on Apache & DSpace projects, in dependence to the number of change request recommendations  $N$ .

#### 4.5 Comparison with Literature

In this segment, the performance of file impact prediction results from the proposed technique of this thesis i.e., VCIA is compared with that from literature. For performance comparison, a subset of file impact prediction papers from literature review chapter is selected. For this comparison purpose, those literature are selected where the performance (measured in terms of Precision and Recall) is measured and reported for varying number of file recommendations. Table 4.8 show the performance values from this study (VCIA) and literature studies (papers). This table is an extension to the file impact literature comparison table reported in [43]. The table presented in this thesis, also reports the performance of VCIA on OSS projects. From the values presented, it is clear that file impact predictions are inherently difficult. This is indicated by lower Precision and Recall values compared to other fields of computer science. Lower values (compared to other disciplines) are true for all techniques from literature as well as this study.

Each of the study report results by applying their respective techniques. “Min” and “Max” row denote the minimum and maximum evaluation values respectively, as reported in the literature study. For two of the studies, only average values are reported in the papers. The figures in bold

represent highest evaluation measure value attained for the number of file recommendations and corresponding evaluation measures (indicated by the respective column headings).

In terms of Precision, the highest values attained is 0.26 for 10 recommendations, which is attained by VCIA. The Precision values from VCIA is consistently higher for the different number of recommendations. Generally, Precision decrease with increasing  $F$  and this is true for all techniques (including this study). Precision determines the fraction of files which are correct in the list of total recommendations. With respect to literature results, this means that on average, VCIA provides higher number of relevant files in the list of file impact recommendations.

In terms of Recall VCIA is outperformed by the techniques of Kagdi et al [45] and Gethers et al. [32]. Overall, the highest Recall is reported by Gethers et al. [32], which is 0.75 for 40 recommendations. In comparison highest Recall obtained from VCIA is 0.67 for 50 recommendations. Generally, Recall improve with increasing number of recommendations across all the studies. Recall determines the percentage of the correct file impact which was retrieved in the list of recommendations. In this respect studies [32] and [45] outperformed VCIA.

Comparing results with literature, it is clear that VCIA generates better results for  $F = 10$  recommendations, which also provide the best trade-off between Precision and Recall. Even though two of the literature techniques provide better Recall, the Precision values are higher for VCIA. This means that VCIA has more correct files in the list of recommendations. In addition, all the other studies with the exception of [21] and [99] utilize additional natural language tokens identified from the code for making file impact prediction. The presence of these tokens relies on adhering and investing in best practices because developers would need to make time to put comments and meaning code entity names. For small teams especially start-ups, where the focus is on quick delivery rather than following best practices. Thus applying these techniques which rely on natural language tokens is difficult. In addition, some of the studies utilize additional data which are difficult to acquire, are not readily available and sometimes require specific IDE plugin for capturing the required data. As examples, Gethers et al. [32] use execution trace while Zanjani et

al. [80] use code interaction data. All these present additional overhead, where small and start-up teams might be reluctant to invest the additional time and effort. The results obtained with VCIA only rely on change request data from issue repositories that are used by all software development companies. The data is readily available, requires no intervention from the developers and is not dependent on the language or construct. Despite using just change request data for file impact prediction, the Precision values are better, and comparable Recall values which are about 0.10 less than best values observed.

Table 4.9 show the performance of VCIA and literature with respect to effort and duration prediction. Two of the studies from literature review section has been selected for this purpose. Majority of the effort and duration prediction from literature utilized classifier models, where performance was reported for different attribute selection. On the other hand, the performance of VCIA is reported for varying number of change requests recommendation  $N$ , as this was the only meaningful attribute which could be varied. Therefore for comparison purpose, only those studies were selected where performance was reported for a different number of change request recommendations. For VCIA as well for the studies presented in Table 4.9, the evaluation measure was MRE accuracy and the error bound  $E$  for VCIA and literature was all set at 25% (0.25) threshold. The value in each cell of each represents the highest accuracy attained by the technique outlined in the literature (row header) for the corresponding number of change requests recommended (column header).

In terms of effort prediction, for all the techniques the prediction accuracy (for both effort and duration prediction) remained largely unchanged for  $N$  greater than 3. VCIA consistently generated better results than [94]. However, Dehghan et al. [27] generated better results compared to VCIA. Although the results from [27] appear significantly better than VCIA, an important to note here is that the technique in the former study was applied on *Work Items* of type *Tasks* and not directly change requests. Similarly, technique outlined in [94] was applied on only *Bug* type change requests. In contrast, VCIA was evaluated on all type of change requests, which could be



Table 4.8: Performance comparison of file impact prediction techniques from literature and proposed approach, shown for different number of file recommendations  $F$ .  $P$  and  $R$  stand for Precision and Recall measures respectively. “-” indicate unavailable evaluation data (adapted from [43]).

Study	Number of file recommendations $F$	10		20		30		40		50	
		P	R	P	R	P	R	P	R	P	R
VCIA	Min	0.07	0.12	0.06	0.16	0.04	0.19	0.04	0.20	0.03	0.21
	Max	<b>0.26</b>	0.44	<b>0.23</b>	0.61	<b>0.22</b>	0.62	<b>0.22</b>	0.64	<b>0.21</b>	0.67
Borg et al. [16]	Min	0.06	0.40	0.06	0.42	0.06	0.43	0.06	0.44	0.06	0.44
	Max	0.07	0.40	0.07	0.50	0.07	0.50	0.07	0.55	0.07	0.55
Gethers et al. [32]	Min	0.06	0.06	0.05	0.12	0.04	0.14	0.04	0.18	-	-
	Max	0.14	0.37	0.10	0.53	0.08	<b>0.64</b>	0.07	<b>0.75</b>	-	-
Kagdi et al. [45]	Min	0.01	0.01	0	0.01	0	0.01	0	0.01	0	0.01
	Max	0.14	<b>0.54</b>	0.10	<b>0.64</b>	0.08	0.70	0.06	0.73	0.05	<b>0.78</b>
Zanjani et al. [97]	Min	0.09	0.18	0.07	0.27	-	-	-	-	-	-
	Max	0.15	0.26	0.12	0.32	-	-	-	-	-	-
Canfora et al. [21]	Min	0.05	0.20	-	-	-	-	-	-	-	-
	Max	0.20	0.40	-	-	-	-	-	-	-	-
Torchiano et al. [93]	Average	0.18	0.23	-	-	-	-	-	-	-	-
Zimmerman et al. [99]	Average	-	0.33	-	-	-	-	-	-	-	-

Table 4.9: Comparison of effort and duration prediction results from literature and VCIA. “-” indicate unavailable evaluation data.

Study	Number of CR recommendations (N)	1	3	5	7	9
VCIA	Effort prediction	<b>0.30</b>	<b>0.44</b>	<b>0.44</b>	<b>0.44</b>	<b>0.44</b>
Dehghan et al. [27]		-	0.69	0.69	0.68	0.68
Weiss et al. [94]		0.17	0.15	0.15	0.16	0.15
VCIA	Duration prediction	0.26	0.32	0.33	0.33	0.33
Pfahl et al. [72]		-	<b>0.39</b>	-	-	-

an important contributing factor to the performance difference.

In terms of duration prediction, there was only study Pfahl et al. [72] which satisfied the selection criteria and the result was only reported for  $N = 3$  recommendations. The performance of [72] was better than VCIA. An important point of difference is that the technique in [72] was applied to data which was segmented into different duration intervals. The duration prediction performance was measured with respect to the duration interval plus error bound. This would mean that the prediction would be correct as long as it fell within in value range of duration interval  $+/-$  error bound 0.25. For VCIA the duration was not segmented into intervals. The segmentation used in [72] provided a larger duration range of acceptance, where as in VCIA it was comparatively less, as the acceptable duration range was the actual duration value  $+/-$  error bound 0.25. Thus, the accuracy values from Pfahl et al. [72] were slightly higher.

Overall, the performance of VCIA for file impact, effort and duration are comparable to those reported in the literature. In some cases, the results are less and the underlying possible reason was discussed in previous paragraphs. One important advantage of VCIA is that it provides file impact, effort, and duration prediction together in an integrated fashion. The techniques from literature discussed in the preceding discussion provide these predictions separately. VCIA on the other hand provide vector impact prediction for change requests i.e., files impacted, effort and duration prediction with respect to CR. The results from very few literature are a little better than VCIA. But overall the benefits of vector impact prediction for enhanced decision making will outweigh this minor limitation. The main objective was to propose a technique which could provide reasonable

vector impact prediction. The performance is largely good (being less only for a few literature) indicates that the results are reasonable, comparable and certainly useful. As future work, the performance aspect of VCIA can be analyzed and improved so that vector impact predictions are even better.

Another point of comparison is the number of data entity analyzed. Table A.1 in Appendix, shows the number of data entity analyzed by VCIA as well as all the studies against which file impact, effort and duration prediction was compared (Table 4.8 and 4.9). All the literature evaluate their techniques on either industry or open source data. VCIA is the only study where the performance was compared with both industry and open source data. VCIA was also evaluated with the highest number of CR compared to the other studies. For prediction, VCIA also requires file and commit data. The only other study which analyzes more data is Kagdi et al. [45]. However, VCIA needs to trace CR with commit, thus it is limited by the amount of CR traceable data available. As Kagdi et al. [45] do not need to exclusively rely on this traceability, they were free to use as much of data as possible. For the case of VCIA, these were the maximum number of data points available for the projects selected.

## 4.6 Summary

This chapter presents the results of two case studies. The first case study was conducted in Brightsquid Inc. The proposed approach was applied to two real-world projects obtained from the industry partner. The second case study was conducted using data from two popular OSS projects, i.e., Apache, and DSpace. Analysis of result show similar trends of performance in both case studies. Looking at the individual techniques, BOW & CO performs best for file impact prediction in both case studies. Combination of BOW with topic BOW & LDA produces best effort estimation results across all projects in both studies. Finally, using simple Bag of Words presentation with textual similarity (BOW) performs best with respect to duration prediction across projects. The case studies and performance evaluation help to attain objective RO3.

# Chapter 5

## VCIA tool support

### 5.1 Introduction

The VCIA tool has been developed as an online application. The tool utilize data stored in CR and Code repositories, Jira and GitHub. VCIA has been integrated into an online analytics dashboard. The dashboard has been developed for Brightsquid Inc, Canada. Figure 5.1 show the home page. The tool is accessible from anywhere without any installation. It does not require development environment setup or configurations for execution. The tool is able to operate without any human intervention, except for the user input. Brightsquid is using the tool as part of their sprint planning and estimation phase. The design and implementation of this tool is geared towards fulfilling objective RO4.

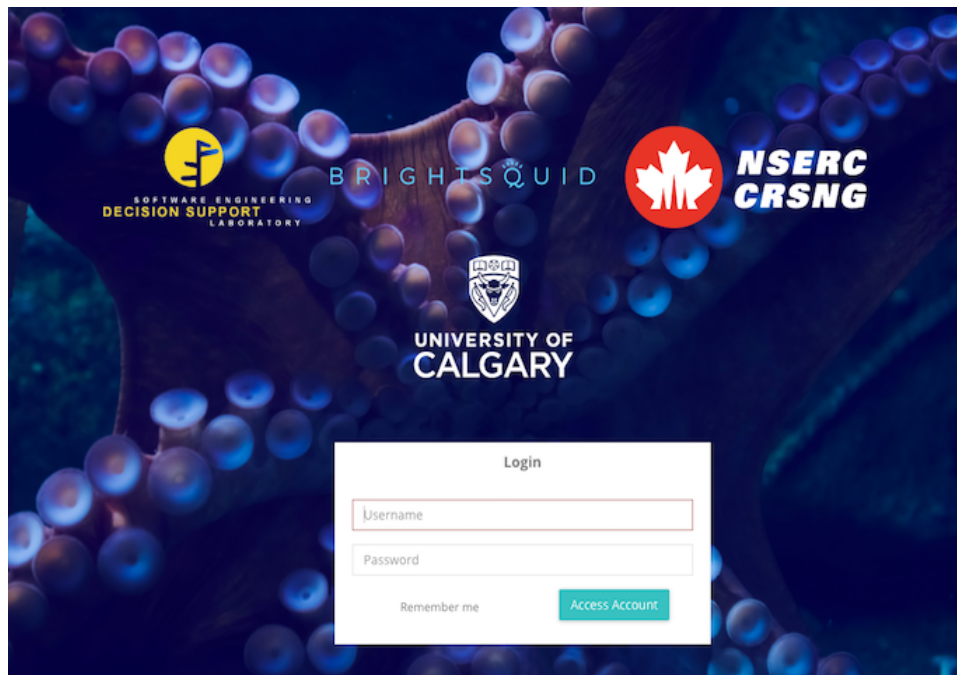


Figure 5.1: Tool homepage

## 5.2 Overview of tool development

The tool use state-of-the-art technologies to generate vector impact prediction service to the user. Table 5.1 highlight the main technologies used in the development of the tool. Entire analytics dashboard is hosted and deployed as web service on the popular *Amazon Web Services (AWS)*, more specifically Amazon Elastic Compute Cloud (Amazon EC2). EC2 is a very popular and secure cloud based web services platform, which make development and deployment of web services quick and efficient. All the tool layers (discussed in next chapter) are hosted on EC2 instance.

For data extraction, GitHub API is used to download data from Brightsquad repository in GitHub. This API is only used in the Data layer preparation. All core functionality of VCIA tool is developed with Java. Popular API are used to achieve various tasks ranging from, data pre-processing to the final result presentation. Core functionality require scanning the repository for finding similar CR. Popular open source search-engine API Apache Lucene is used to index and query data entities for finding suitable matches, with respect to CR input. These technologies are primarily used in Application Layer.

HTML, Javascript, JSP are jointly used to develop the web pages to take user inputs and return outputs back. Some portion of the tool require graph generation from the output of the core program. This is achieved with Python scripts integrated into the tool. Together they form the front-end of the tool and also the entire dashboard, primarily used in the Presentation Layer.

Table 5.1: List of tools and technologies used for VCIA tool development

<b>Tools &amp; Technology</b>	<b>Layer</b>	<b>Usage</b>
GitHub API	Data	Data extraction
Java	Application	Application development
HTML, Javascript, JSP	Presentation	Develop webpage
Python	Presentation	Graph Generation
Amazon Web Services	N/A	Tool Deployment

## 5.3 Architecture

VCIA has a multi-layer architecture for generating vector impact prediction. The tool follows a three layered architecture, which is shown in Figure 5.2. The following segment discuss each layer.

- **Data Layer:** The layer handles data importing from project management tools i.e., GitHub and Jira. The data is downloaded through a separate secure stand-alone program and imported into the tool. The tool store the data in the data layer. *Code data* in Figure 5.2 refer to data related to code entities, which are retrieved from GitHub. The data is downloaded via standalone program which uses the GitHub API. Standalone program is a Java based application, that downloads the data from the servers with credentials created by the company. Brightsquid maintains a number of code repositories in GitHub, which are maintained as separate projects. From these repositories, data from all active projects are retrieved at regular weekly intervals. Subsequently, the data layer is updated with recent data. Change Request (CR) data is collected from the Mail project of in Jira. It is currently the only live CR project being maintained, under which changes are managed in multiple products. CR data is fetched via the online interface provided by Jira.
- **Application Layer:** All processing and calculations take place in this layer. This layer receive a JSON object as input, which also initiates processing for this layer. Application layer generates results for two modules (mentioned in next chapter). For the first module *Ripple Effect Analysis*, the data is first pre-processed via different textual cleaning techniques implemented with different class files. For text preparations and cleaning, numerous popular API and regular expression are used. In particular, for stemming text, an OSS Java implementation of Porter Stemmer algorithm is used [73]. CR to code traceability is also established in this layer. Traceability is primarily established by using regular expressions to look for CR ID

in commit messages. One of the key preparation task is Duration value resolution because this value is not readily available. This is done by individually looking at each CR for the opening and closing dates. Similarly effort values are normalized and converted to *hours* from *seconds*, this is because the online provide effort values in seconds. All these comprise of the processing that need to be completed before the vector impact can be predicted. Once all the data is ready, impact prediction can be completed. Impact prediction processing begins when the user has provided an input, based on which the vector impact of the corresponding change request is generated. Apache Lucene is used to index and query the data entities for finding suitable matches, with respect to input CR. Based on the results of similarity, other relevant details are retrieved from Data layer (CR and Code data), for packaging final output. The results are passed back to the presentation layer as JSON object.

Processing for the second module *Commit Traceability Analysis* involves looking at commit traceability of 5 most active GitHub projects. A Java program analyze all commit messages to find presence of CR ID. If it is found, then the commit is treated as *traceable*. The result of this module are the ratio of the commits, distributed into multiple time buckets. More details of the input and output are presented in next segment. The results are sent to the presentation layer for graphical representation. Similar to the first module, inputs and outputs are both in JSON formats, received and sent back to the Presentation Layer respectively.

- **Presentation Layer:** The presentation layer is responsible for accepting user input and display output back to users. For the *Ripple Effect Analysis* module, three set of outputs are generated, in response to a change request. The first output is a list of *Impacted Files*. The second output is the list of *Similar Issues*. The last output are predicted values of effort and duration, for the input CR. Overall, they provide recommendation of the file impact with effort and duration that might be

required to implement the query CR. In this layer, the JSON objects returned from Application Layer are transformed into HTML tables using a combination of HTML, Javascript, CSS. A set of Javascript functions also allow the user to sort the results according to different file attributes.

In the second module *Commit Traceability Analysis*, graphs are generated using values from JSON objects, returned by Application Layer. Graphs are generated using Python scripts, which are embedded in the Application Layer. Using a combination of HTML, Javascript, CSS they are displayed in this layer to the users.

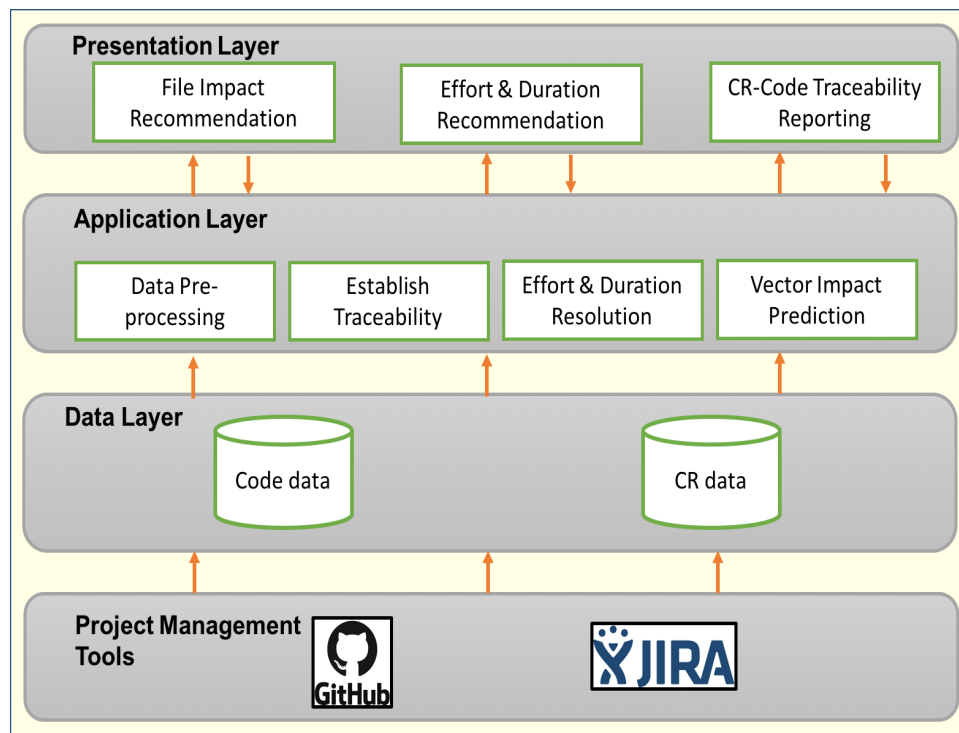


Figure 5.2: Tool architecture layers.

## 5.4 Tool modules

The tool has two main modules, *Ripple Effect Analysis* and *Commit Traceability Analysis*. Each is discussed in the following sub-section.



### 5.4.1 Ripple Effect Analysis

This is the center piece of the analytics dashboard and also the main module of this tool. This module has been developed following the design of VCIA. This analysis looks at the past change history (i.e. JIRA tickets) to generate vector impact of a given change request. Throughout this segment and also in the tool, Issue and CR refer to the same thing. The input and output, with relevant screen-shots are presented below.

**Module Input:** The input are two string values representing *Issue Summary* and *Issue Description*, which are analogous to CR summary and description respectively. Figure 5.3 show the input fields of the tool. It is not mandatory for the inputs to resemble a JIRA ticket Summary and Description, it can also be any group of textual term(s). The search for similar CR is conducted based on these terms entered by user. If similar issues with traceable file impacts are found, then they are displayed to the user.

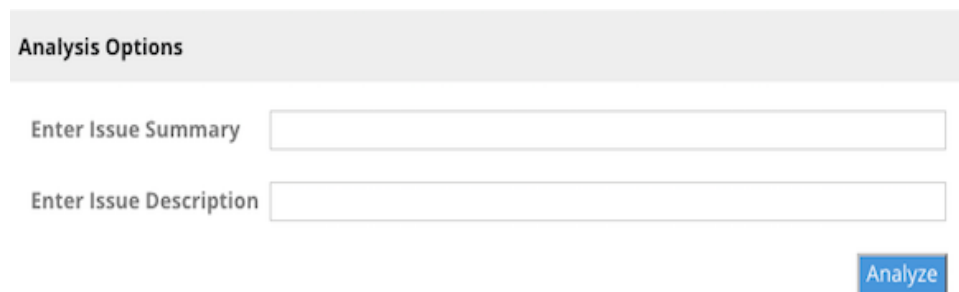
The image shows a web interface for the 'Ripple Effect Analysis' module. At the top, there is a grey header bar with the text 'Analysis Options'. Below this, there are two input fields. The first field is labeled 'Enter Issue Summary' and the second is labeled 'Enter Issue Description'. Both fields are empty text boxes. To the right of these fields, there is a blue button with the text 'Analyze' in white.

Figure 5.3: Input of *Ripple Effect Analysis* module for vector impact recommendation.

**Results & Output:** Three sets of output are returned from this module. The first output is the list of *Impacted Files*. Figure 5.4 is a sample output, showing list of impacted files. The list of *Impacted Files*, can be sorted by three attributes/columns i.e. *Similarity*, *Days Since Last Modified* and *Total Churn*. The list of similar issues are sorted according to similarity, by default. For sorting by any one of the three attributes, users can click on the respective column names. *Similarity* represents the textual

similarity value of the CR which impacted the file. *Days Since Last Modified* shows the number of days ago that the file was changed. Finally, *Total Churn* is the total number of lines added or deleted to the file. For each file, *CoupledFiles* are also shown i.e., files which changed together in the past.

Results

Impacted Files

Rank	Similarity	Days Since Last Modified	Total Churn	File Name	CoupledFiles
1	1	66	8615		
2	1	105	395		

Figure 5.4: Output of *Ripple Effect Analysis* showing *Impacted Files* recommendation (sensitive values hidden to preserve industry confidentiality).

The second output are the predicted values of effort and duration, with respect to the CR input. Figure 5.5 show sample prediction values generated for the user input. The values have been calculated by considering three most similar existing CR.

Analysis

Predicted Effort(hours) 10.68

Predicted Duration(days) 4

Figure 5.5: Output of *Ripple Effect Analysis* showing predicted effort and duration value recommendation.

The third output of this module is the list of *Similar Issues*. These are the CR which are textually similar to the query CR entered by the user. Figure 5.6 show sample

output of *Similar Issues* generated by this module. From this list, the previous two outputs were generated. Each similar CR is described by multiple attributes, which are: *Issue ID*, *Issue Summary*, *Issue Description*, *Issue Severity*, *Actual Effort Required(hrs)*, *Org. Effort Estimate(hrs)* and *Duration*.

Results

Similar Issues ▾

Rank	Similarity Value	Act. Effort Required(hrs)	Issue Summary	Issue Severity	Issue Duration(days)	Issue Id	Org. Effort Estimate(hrs)	Issue Description
1	1	9.02			8		0	

Figure 5.6: Output of *Ripple Effect Analysis* showing *Similar Issues* recommendation (sensitive values hidden to preserve industry confidentiality).

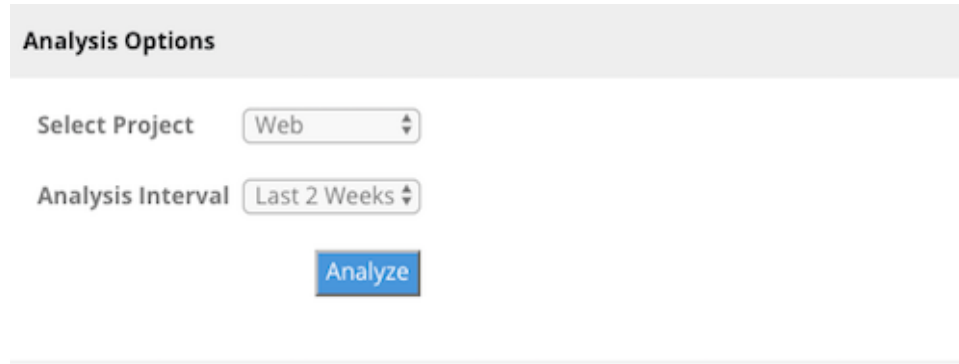
Both *Similar Issues* and *Impacted Files* are shown as HTML tables. Users can select between the two tables using the dropdown boxes. Each issue is described by multiple columns as shown in the tables.

#### 5.4.2 Commit Traceability Analysis

This sub-module analyze the traceability of CR stored as JIRA tickets, with code commits in GitHub. This module analyze traceability for five active GitHub projects of Brightsquid. The user select the projects from *Select Project* drop down. The tool identifies the percentage of traceable commits from GitHub commit data, by tracing the presence of JIRA ticket/CR ID in the commit messages. Based on *Analysis Interval* chosen, traceability information is displayed for the last two weeks, month, year or entire project life-time.

**Module Input:** This module requires two inputs from the user in *Analysis Options*. The first input is selection of the project from *Select Project*, as mentioned before five active

projects are tracked. The second input is *Analysis Interval*. This input set time line of the analysis and users can select one of the four options: "Last 2 Weeks", "Last 4 Weeks", "Last Year" and "Project Life". The last option displays traceability per month starting from project initiation. Figure 5.7 shows the input selection options for this module.



The screenshot shows a web interface for the 'Analysis Options' module. At the top, there is a header 'Analysis Options'. Below it, there are two dropdown menus. The first is labeled 'Select Project' and has 'Web' selected. The second is labeled 'Analysis Interval' and has 'Last 2 Weeks' selected. Below these menus is a blue button labeled 'Analyze'.

Figure 5.7: Input screen of *Commit Traceability Analysis* module.

**Results & Output:** The output of this module is a graph showing the traceability of the project and analysis interval chosen by the user. Figure 5.8 show sample output generated from this module, for four week interval. For each week, both the total and traceable number of commits are displayed on the left and right y-axis respectively. The line shows the percentage of traceable commits.

## 5.5 Summary

This chapter elaborates on the tool, which has been designed and implemented on the work presented in this thesis. The tool has two main modules. *Ripple Effect Analysis* is the center piece of the tool, it takes CR as input and generates vector impact of the CR as the output. This module has been designed followed VCIA technique outlined previously. The design and implementation of a module which can predict vector impact targets objective RO4.1. The second module *Commit Traceability Analysis* acts as supporting component to the former module, it tracks the percentage

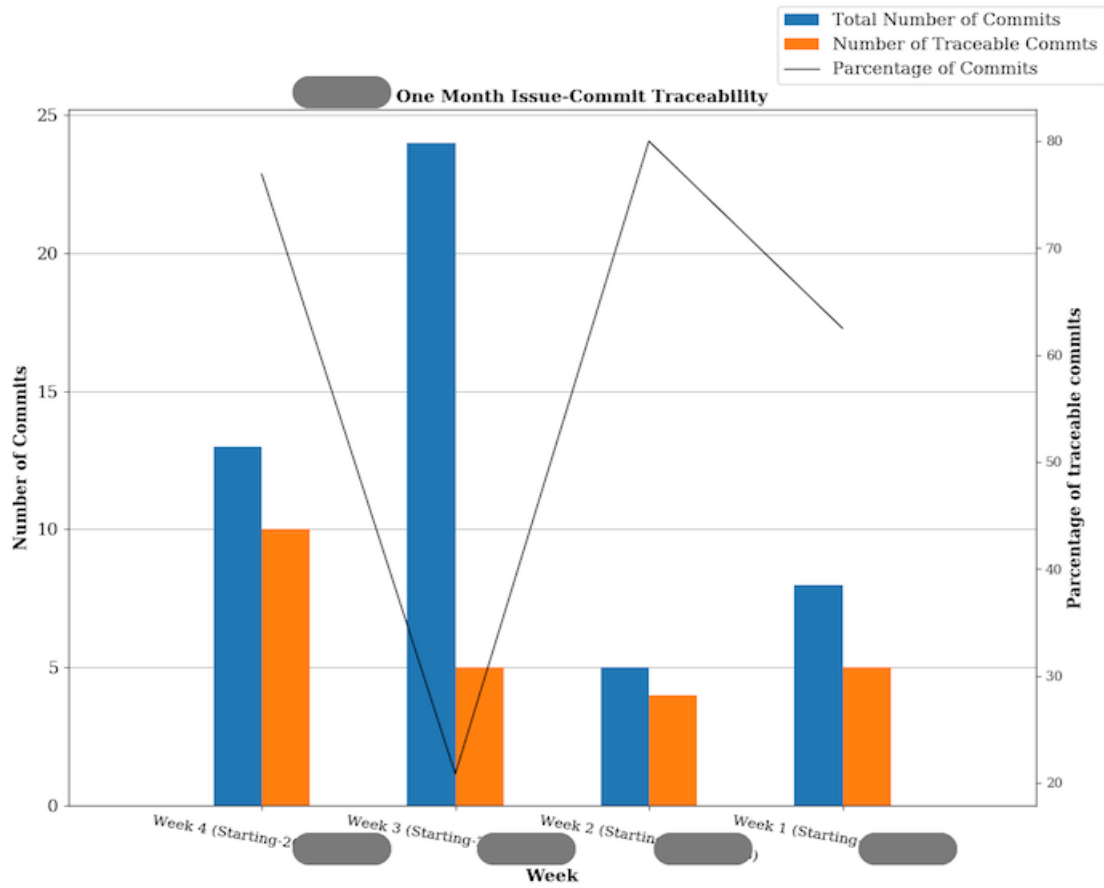


Figure 5.8: Sample output of *Commit Traceability Analysis* (sensitive values hidden to preserve industry confidentiality).

of code commits which are traceable to JIRA tickets. The tool is hosted as an online service that does not require any installation or setup and can generate output without human intervention. The tool is integrated into the analytics dashboard of software company Brightsquid. This helps to fulfill objective RO4.2. This chapter sheds light on the architecture along with the tools and technologies integrated for tool operation. Each of the key modules are outlined with descriptions and screen-shots of the major input and output components.

## Chapter 6

### Summary & Future research

This chapter serves to summarize this entire thesis. The first section discusses the key contributions. The second sections discuss the limitations and Applicability of the results. The final section outlines the possible future directions of this research endeavor.

#### 6.1 Summary of contributions

The key contributions of this thesis are discussed in the subsequent subsections.

##### 6.1.1 VCIA approach for predicting files impacted, effort and duration of change request

Quick and efficient reaction to software change request is of paramount importance. This thesis proposes a three-directional framework that can accept change request as input and generate files impacted, effort and duration value as output. Three textual similarity based techniques for file prediction are presented. Two of the techniques defined for file impact prediction are extended for effort and duration prediction. Through case study validation it was possible to show BOW & CO performs best for file impact prediction in both case studies. Combination of BOW with topic BOW & LDA produce best effort estimation results across all projects in both studies. Finally, using simple Bag of Words presentation with textual similarity (BOW) performs best with respect to duration prediction in both case studies.

##### 6.1.2 Application and validation with industrial case study

The approach outlined in this thesis has been integrated into an analytics dashboard for Brightsquid, a Canadian software company. The VCIA approach is being utilized by the company, and the solutions are acceptable to them. The broadening of textual similarity towards predicting not

only the impacted files, but also effort and duration required to implement the change request, enhance the productivity of the technique and increases industrial relevance. Until now all estimation relied on ad-hoc and manual decision making, resulting in error and poor accuracy. Using the new predictions without incurring additional cost is lucrative for Brightsquid's project management. It reinforces their strategy towards the development of Minimum Viable Products. Brightsquid considers the recommendations as important inputs to their human decision-making process regarding the selection and quantity of change request that should be handled in different releases.

### 6.1.3 Tool development for vector impact prediction

VCIA has been integrated into an online analytics dashboard. The dashboard has been developed for Brightsquid Inc, Canada. The tool is accessible from anywhere and does not need to be installed as an IDE extension such as Eclipse. The tool does not require environment setup or configuration to run. The developed tool can operate without any human intervention. Brightsquid is using the tool as part of their change management process and is satisfied with the results.

## 6.2 Limitations & Applicability

VCIA relies heavily on the traceability between code entities and CR to make file impact predictions. Also, the availability of a good chunk of effort data is also critical. From software repositories, VCIA needs to discover a relatively high number of traceable CR which also have effort estimates available. This is vital for providing vector predictions. Unfortunately, good traceability and effort estimation logging practices are normally not followed by all organizations. Organizations which follow proper development processes and guidelines, tend to follow these good practice only. Thus VCIA is suitable for scenarios where enough traceability and effort data are available.

The proposed approach requires some parameter values and threshold settings to operate effectively. The right or wrong selection of these parameters and threshold can generate very good or bad results. Current threshold and parameter values are defined based on experimentation and

experience. However, these values might be context specific and can cause VCIA to perform less effectively with other projects. To overcome this problem, VCIA would first need to be tuned with the optimum setting before it can be applied on other projects.

The primary objective of this thesis was to generate vector impact for a change request. The motivation was to aid decision makers with recommendations on the vector impact of change. However, there was less focus on improving the overall performance of the proposed approach. The technique presented in this thesis can, therefore, be a good starting point for enhancing vector predictions of change.

### 6.3 Future research scope

The proposed VCIA approach and the associated tool can be further improved. The following section concludes this thesis with discussion of the possible future research directions.

1. **File impact prediction at finer levels of granularity:** VCIA predicts code impact at the file level using only CR textual similarity. A good direction for future research would be to predict impact at a more finer level i.e., methods or even lines of code. In order to achieve this, textual content of the code entities may be leveraged such as comments, identifiers, various entity names etc. In addition, additional code attributes may be leveraged for generating better prediction results.
2. **Analysis on more project data:** Further validation and analysis of VCIA performance is required. This can be done by applying VCIA on additional project data and monitoring performance. This would provide further insight into the capability of the proposed method towards assisting change impact decision making.
3. **Inclusion of additional CR attributes:** Currently VCIA utilizes textual CR content i.e., *Title* and *Description*. In the future additional CR attributes can be included into the prediction technique.



4. **Evaluation measure:** The current evaluation measure only considers the presence or absence of the file. An additional measure such as *MAP* can be used, which also considers the position of predicted files in the list of generated recommendations.
5. **Analysis with varied parameter setting:** Additional result generation and analysis of VCIA performance can be done by varying key parameters and threshold values. This will serve to evaluate robustness and stability of the solution approach.
6. **Order of files:** The file impact prediction procedure does not consider the ordering of files. Ordering of the files maybe important towards the final results. As future, the impact of the order of files and change requests can be taken into account.
7. **Tool live data update:** Providing the tool with the ability to update data automatically. Currently, live data update is disabled due to some technical issues. Fixing this feature would make the tool more independent.

## Appendix A

### Additional figures, tables from experimental results and copyright form

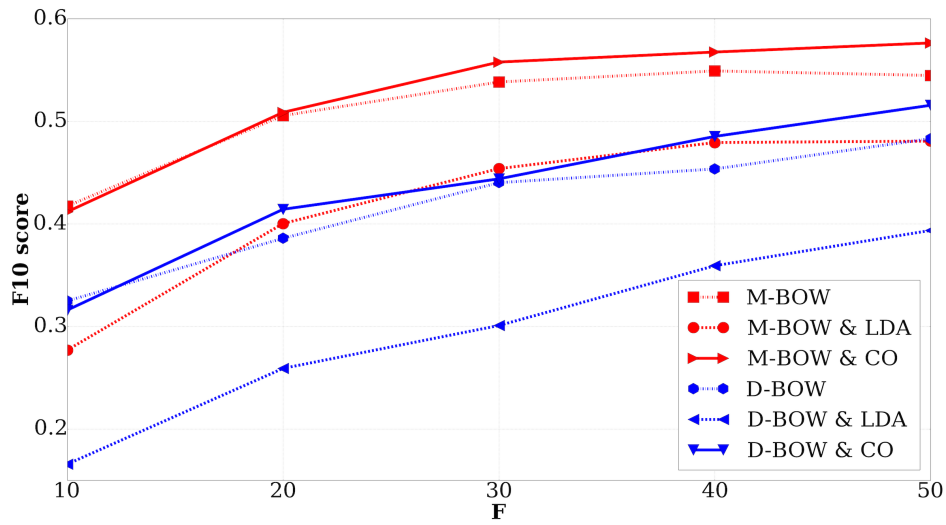


Figure A.1: F10 scores of file impact prediction by applying the three proposed techniques on industry projects Mail (M) and Dental (D), in dependence to the number of file recommendations  $F$  [43].

#### MAIL PROJECT: DISTRIBUTION OF CHANGE REQUEST TYPES

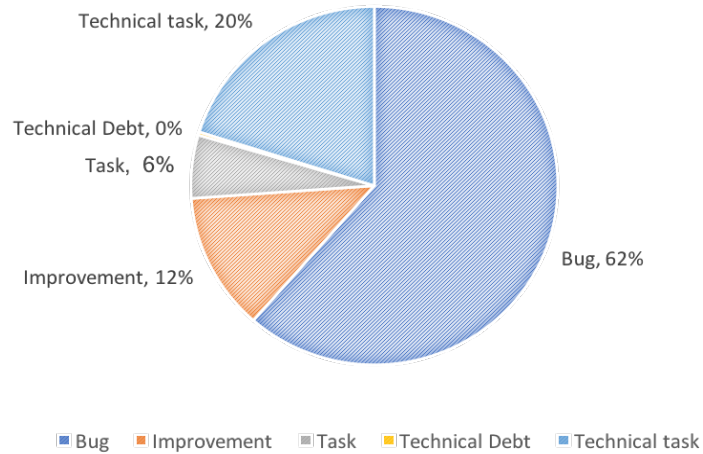


Figure A.2: Ratio of change request types for Mail project.

#### DENTAL PROJECT: DISTRIBUTION OF CHANGE REQUEST TYPE

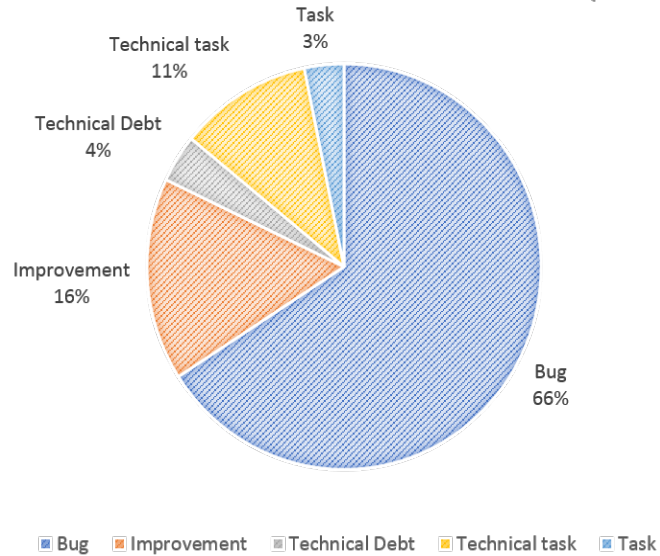


Figure A.3: Ratio of change request types for Dental project.

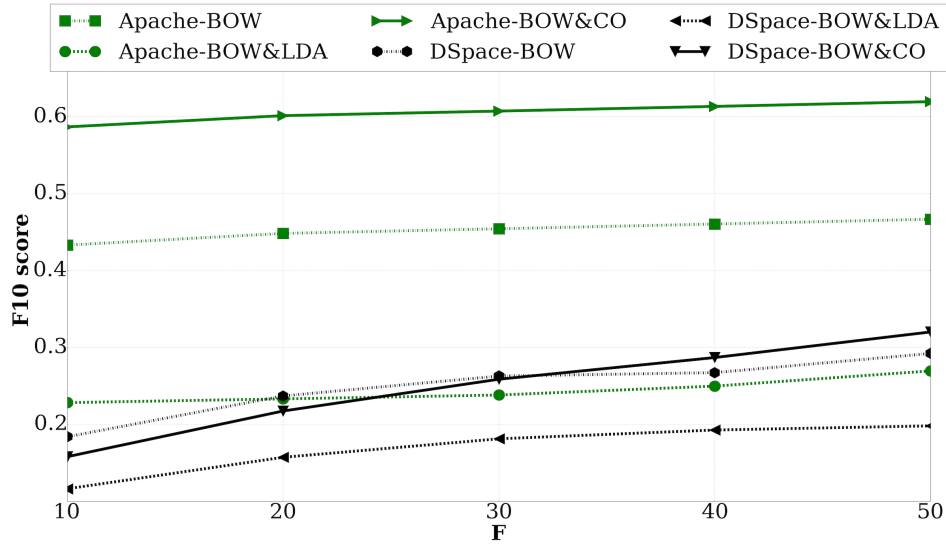


Figure A.4: F10 scores of file impact prediction by applying the three proposed techniques on industry projects Apache and DSpace, in dependence to the number of file recommendations  $F$ .

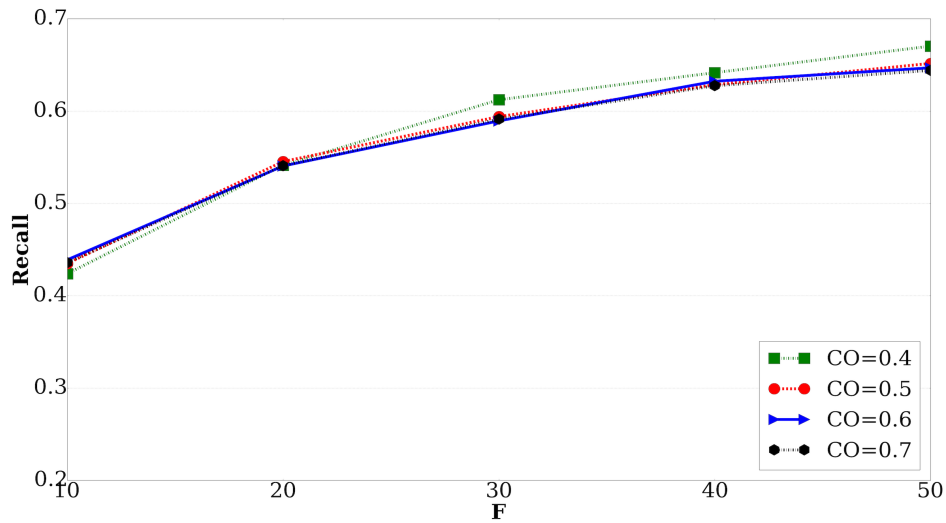


Figure A.5: Recall of file impact prediction for different values of  $CO$  threshold, by applying the BOW & CO on Mail, in dependence to the number of file recommendations  $F$ .

Table A.1: Number of entity analyzed in VCIA and literature studies.

Study	Number of projects	Data source	Number of entity (entity name)
VCIA	4	Industry & Open source	<b>2386</b> <b>(change request)</b>
			6546 (commit)
			3811 (file)
Borg et al. [16]	2	Industry	2000 (non-code entities)
Gethers et al. [32]	4	Open source	277 (change request)
			3588 (file)
Kagdi et al. [45]	6	Open source	<b>16551</b> <b>(file)</b>
			<b>11377</b> <b>(commit)</b>
Zanjani et al. [97]	1	Open source	3727 (commit)
Canfora et al. [21]	3	Open source	1377 (change request)
			1744 (file)
Torchiano et al. [93]	5	Open source	6444 (file)
Zimmerman et al. [99]	8	Open source	4000 (transaction)
Weiss et al. [94]	1	Open source	2125 (change request)
Dehghan et al. [27]	2	Industry	9417 (work item)
Pfahl et al. [72]	1	Industry	567 (change request)

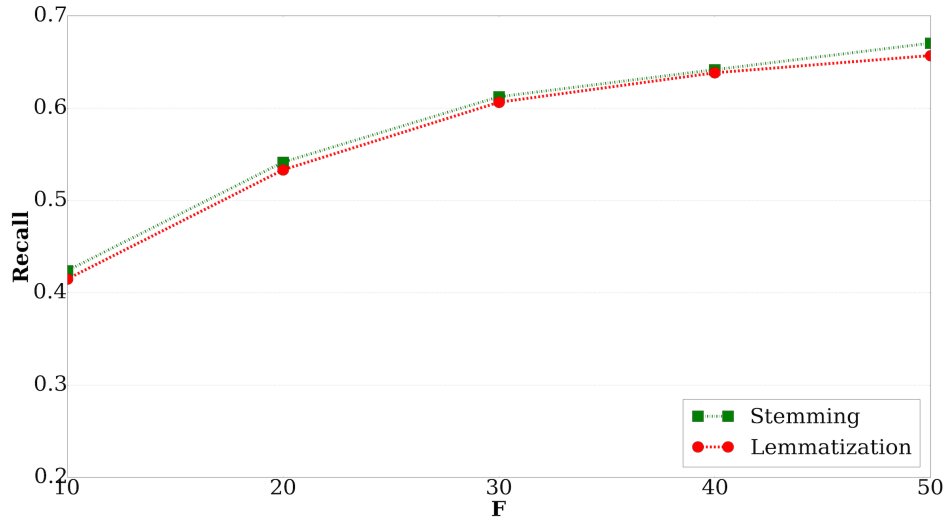


Figure A.6: Recall of file impact prediction for *Stemming* and *Lemmatization* operations, by applying the BOW & CO on Mail, in dependence to the number of file recommendations  $F$ .

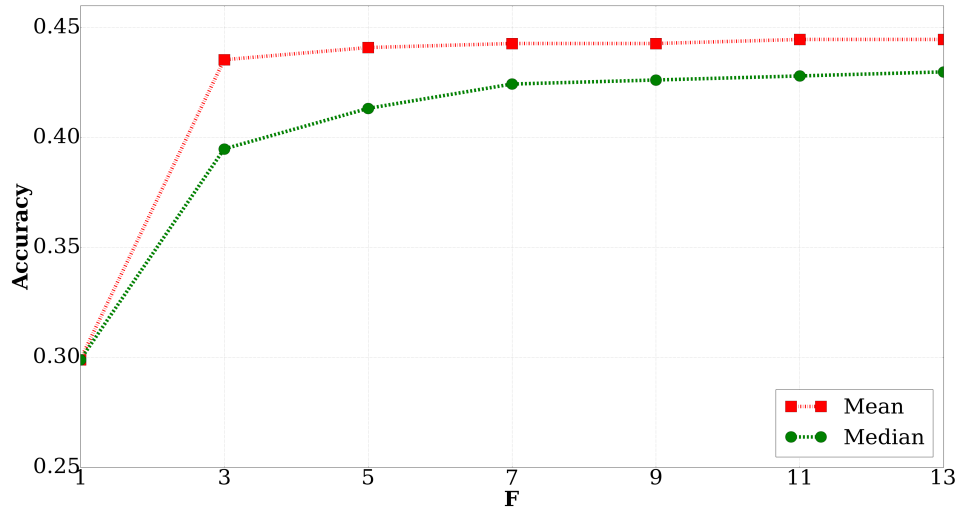


Figure A.7: Accuracy of effort prediction for *Mean Median* value calculation, by applying BOW & LDA on Mail, in dependence to the number of change request recommendations  $N$ .

## IEEE COPYRIGHT AND CONSENT FORM

To ensure uniformity of treatment among all contributors, other forms may not be substituted for this form, nor may any wording of the form be changed. This form is intended for original material submitted to the IEEE and must accompany any such material in order to be published by the IEEE. Please read the form carefully and keep a copy for your files.

**Predicting the Vector Impact of Change - An Industrial Case Study at Brightsquad**

**Shaikh Jeeshan Kabeer**

**2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement**

### COPYRIGHT TRANSFER

The undersigned hereby assigns to The Institute of Electrical and Electronics Engineers, Incorporated (the "IEEE") all rights under copyright that may exist in and to: (a) the Work, including any revised or expanded derivative works submitted to the IEEE by the undersigned based on the Work; and (b) any associated written or multimedia components or other enhancements accompanying the Work.

### GENERAL TERMS

1. The undersigned represents that he/she has the power and authority to make and execute this form.
2. The undersigned agrees to indemnify and hold harmless the IEEE from any damage or expense that may arise in the event of a breach of any of the warranties set forth above.
3. The undersigned agrees that publication with IEEE is subject to the policies and procedures of the [IEEE PSPB Operations Manual](#).
4. In the event the above work is not accepted and published by the IEEE or is withdrawn by the author(s) before acceptance by the IEEE, the foregoing copyright transfer shall be null and void. In this case, IEEE will retain a copy of the manuscript for internal administrative/record-keeping purposes.
5. For jointly authored Works, all joint authors should sign, or one of the authors should sign as authorized agent for the others.
6. The author hereby warrants that the Work and Presentation (collectively, the "Materials") are original and that he/she is the author of the Materials. To the extent the Materials incorporate text passages, figures, data or other material from the works of others, the author has obtained any necessary permissions. Where necessary, the author has obtained all third party permissions and consents to grant the license above and has provided copies of such permissions and consents to IEEE.

**You have indicated that you DO NOT wish to have video/audio recordings made of your conference presentation under terms and conditions set forth in "Consent and Release."**

BY TYPING IN YOUR FULL NAME BELOW AND CLICKING THE SUBMIT BUTTON, YOU CERTIFY THAT SUCH ACTION CONSTITUTES YOUR ELECTRONIC SIGNATURE TO THIS FORM IN ACCORDANCE WITH UNITED STATES LAW, WHICH AUTHORIZES ELECTRONIC SIGNATURE BY AUTHENTICATED REQUEST FROM A USER OVER THE INTERNET AS A VALID SUBSTITUTE FOR A WRITTEN SIGNATURE.

Shaikh Jeeshan Kabeer

Signature

18-09-2017

Date (dd-mm-yyyy)

### Information for Authors

#### AUTHOR RESPONSIBILITIES

The IEEE distributes its technical publications throughout the world and wants to ensure that the material submitted to its publications is properly available to the readership of those publications. Authors must ensure that their Work meets the requirements as stated in section 8.2.1 of the IEEE PSPB Operations Manual, including provisions covering originality, authorship, author responsibilities and author misconduct. More information on IEEE's publishing policies may be found at [http://www.ieee.org/publications\\_standards/publications/rights/authorrightsresponsibilities.html](http://www.ieee.org/publications_standards/publications/rights/authorrightsresponsibilities.html). Authors are advised especially of IEEE PSPB Operations Manual section 8.2.1.B12: "It is the responsibility of the authors, not the IEEE, to determine whether disclosure of their material requires the prior consent of other parties and, if so, to obtain it." Authors are also advised of IEEE PSPB Operations Manual section 8.1.1B: "Statements and opinions given in work published by the IEEE are the expression of the authors."

#### **RETAINED RIGHTS/TERMS AND CONDITIONS**

- Authors/employers retain all proprietary rights in any process, procedure, or article of manufacture described in the Work.
- Authors/employers may reproduce or authorize others to reproduce the Work, material extracted verbatim from the Work, or derivative works for the author's personal use or for company use, provided that the source and the IEEE copyright notice are indicated, the copies are not used in any way that implies IEEE endorsement of a product or service of any employer, and the copies themselves are not offered for sale.
- Although authors are permitted to re-use all or portions of the Work in other works, this does not include granting third-party requests for reprinting, republishing, or other types of re-use. The IEEE Intellectual Property Rights office must handle all such third-party requests.
- Authors whose work was performed under a grant from a government funding agency are free to fulfill any deposit mandates from that funding agency.

#### **AUTHOR ONLINE USE**

- **Personal Servers.** Authors and/or their employers shall have the right to post the accepted version of IEEE-copyrighted articles on their own personal servers or the servers of their institutions or employers without permission from IEEE, provided that the posted version includes a prominently displayed IEEE copyright notice and, when published, a full citation to the original IEEE publication, including a link to the article abstract in IEEE Xplore. Authors shall not post the final, published versions of their papers.
- **Classroom or Internal Training Use.** An author is expressly permitted to post any portion of the accepted version of his/her own IEEE-copyrighted articles on the author's personal web site or the servers of the author's institution or company in connection with the author's teaching, training, or work responsibilities, provided that the appropriate copyright, credit, and reuse notices appear prominently with the posted material. Examples of permitted uses are lecture materials, course packs, e-reserves, conference presentations, or in-house training courses.
- **Electronic Preprints.** Before submitting an article to an IEEE publication, authors frequently post their manuscripts to their own web site, their employer's site, or to another server that invites constructive comment from colleagues. Upon submission of an article to IEEE, an author is required to transfer copyright in the article to IEEE, and the author must update any previously posted version of the article with a prominently displayed IEEE copyright notice. Upon publication of an article by the IEEE, the author must replace any previously posted electronic versions of the article with either (1) the full citation to the IEEE work with a Digital Object Identifier (DOI) or link to the article abstract in IEEE Xplore, or (2) the accepted version only (not the IEEE-published version), including the IEEE copyright notice and full citation, with a link to the final, published article in IEEE Xplore.

**Questions about the submission of the form or manuscript must be sent to the publication's editor.**

**Please direct all questions about IEEE copyright policy to:**

**IEEE Intellectual Property Rights Office, [copyrights@ieee.org](mailto:copyrights@ieee.org), +1-732-562-3966**



## Bibliography

- [1] H. Abdeen, K. Bali, H. Sahraoui, and B. Dufour. Learning dependency-based change impact predictors using independent change histories. *Information and Software Technology*, 67:220–235, 2015.
- [2] W. Abdelmoez, M. Kholief, and F. M. Elsalmy. Bug fix-time prediction model using naive bayes classifier. In *2012 22nd International Conference on Computer Theory and Applications (ICCTA)*, pages 167–172, 2012.
- [3] M. K. Abdi, H. Lounis, and H. Sahraoui. Predicting change impact in object-oriented applications with bayesian networks. In *2009 33rd Annual IEEE International Computer Software and Applications Conference*, volume 1, pages 234–239, 2009.
- [4] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB '94*, pages 487–499, 1994.
- [5] S. M. D. A. Alam, M. R. Karim, D. Pfahl, and G. Ruhe. Comparative analysis of predictive techniques for release readiness classification. In *2016 IEEE/ACM 5th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE)*, pages 15–21, 2016.
- [6] A. Albrecht. Measuring Application Development Productivity. In *IBM Application Development Symp.*, pages 83–92, 1979.
- [7] A. AlSanad and A. Chikh. *Reengineering of Software Requirement Specification*, pages 95–111. Springer International Publishing, Cham, 2014.
- [8] B. Andam, A. Burger, T. Berger, and M. R. Chaudron. FLORIDA: Feature LOcatIon DASHboard for Extracting and Visualizing Feature Traces. *Proceedings of the Eleventh Inter-*

- national Workshop on Variability Modelling of Software-intensive Systems*, pages 100–107, 2017.
- [9] R. S. Arnold and S. A. Bohner. Impact analysis-towards a framework for comparison. In *Proceedings of 1993 Conference on Software Maintenance CSM-93*, pages 292–301, 1993.
- [10] S. Assar, M. Borg, and D. Pfahl. Using text clustering to predict defect resolution time: a conceptual replication and an evaluation of prediction accuracy. *Empirical Software Engineering*, 21(4):1437–1475, 2016.
- [11] I. R. M. Association. *Business Intelligence: Concepts, Methodologies, Tools, and Applications*. IGI Global, Hershey, PA, USA, 1st edition, 2015.
- [12] R. A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.
- [13] A. Barua, S. W. Thomas, and A. E. Hassan. What are developers talking about? an analysis of topics and trends in stack overflow. *Empirical Softw. Engg.*, 19(3):619–654, June 2014.
- [14] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, 2003.
- [15] B. W. Boehm, Clark, Horowitz, Brown, Reifer, Chulani, R. Madachy, and B. Steece. *Software Cost Estimation with Cocomo II with Cdrom*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2000.
- [16] M. Borg, K. Wnuk, B. Regnell, and P. Runeson. Supporting change impact analysis using a recommendation system: An industrial case study in a safety-critical context. *IEEE Transactions on Software Engineering*, 43(7):675–700, 2017.
- [17] L. Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, Oct. 2001.

- [18] J. Buckner, J. Buchta, M. Petrenko, and V. Rajlich. JRipples: A tool for program comprehension during incremental change. *Proceedings - IEEE Workshop on Program Comprehension*, pages 149–152, 2005.
- [19] H. Cai and R. Santelices. A comprehensive study of the predictive accuracy of dynamic change-impact analysis. *J. Syst. Softw.*, 103(C):248–265, May 2015.
- [20] G. Canfora and L. Cerulo. Impact analysis by mining software and change request repositories. In *Proceedings of the 11th IEEE International Software Metrics Symposium, METRICS '05*, pages 29–, 2005.
- [21] G. Canfora and L. Cerulo. Fine grained indexing of software repositories to support impact analysis. In *Proceedings of the 2006 International Workshop on Mining Software Repositories, MSR '06*, pages 105–111, 2006.
- [22] Y. C. Cavalcanti, P. A. da Mota Silveira Neto, I. d. C. Machado, T. F. Vale, E. S. de Almeida, and S. R. d. L. Meira. Challenges and opportunities for software change request repositories: a systematic mapping study. *Journal of Software: Evolution and Process*, 26(7):620–653, 2014.
- [23] T.-H. Chen, S. W. Thomas, and A. E. Hassan. A survey on the use of topic models when mining software repositories. *Empirical Software Engineering*, 21(5):1843–1919, 2016.
- [24] M. Chochlov, M. English, and J. Buckley. A historical, textual analysis approach to feature location. *Information and Software Technology*, 88:110–126, 2017.
- [25] J. Cleland-Huang, R. Settими, C. Duan, and X. Zou. Utilizing supporting evidence to improve dynamic requirements traceability. In *13th IEEE International Conference on Requirements Engineering (RE'05)*, pages 135–144, 2005.
- [26] R. Colomo-Palacios, C. Casado-Lumbreras, P. Soto-Acosta, F. J. García-Peñalvo, and E. To-var. Project managers in global software development teams: A study of the effects on

- productivity and performance. *Software Quality Journal*, 22(1):3–19, 2014.
- [27] A. Dehghan, K. Blincoe, and D. Damian. A hybrid model for task completion effort estimation. In *Proceedings of the 2Nd International Workshop on Software Analytics*, SWAN 2016, pages 22–28, 2016.
- [28] B. Dit, M. Wagner, S. Wen, W. Wang, M. Linares-Vásquez, D. Poshyvanyk, and H. Kagdi. ImpactMiner: a tool for change impact analysis. *Companion Proceedings of the 36th International Conference on Software Engineering - ICSE Companion 2014*, pages 540–543, 2014.
- [29] A. N. Duc and P. Abrahamsson. *Minimum Viable Product or Multiple Facet Product? The Role of MVP in Software Startups*, pages 118–130. Springer International Publishing, Cham, 2016.
- [30] G. R. Finnie, G. E. Wittig, and J. M. Desharnais. *Estimating software development effort with case-based reasoning*, pages 13–22. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.
- [31] D. D. Galorath and M. W. Evans. *Software Sizing, Estimation, and Risk Management*. Auerbach Publications, Boston, MA, USA, 2006.
- [32] M. Gethers, B. Dit, H. Kagdi, and D. Poshyvanyk. Integrated impact analysis for managing software changes. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE ’12, pages 430–440, 2012.
- [33] M. Gethers and D. Poshyvanyk. Using relational topic models to capture coupling among classes in object-oriented software systems. In *Software Maintenance (ICSM), 2010 IEEE International Conference on*, pages 1–10, 2010.
- [34] E. Giger, M. Pinzger, and H. Gall. Predicting the fix time of bugs. In *Proceedings of the 2Nd International Workshop on Recommendation Systems for Software Engineering*, RSSE ’10, pages 52–56, 2010.

- [35] P. A. Gonzalez. Applying knowledge modelling and case-based reasoning to software reuse. *IEE Proceedings - Software*, 147(5):169–177, 2000.
- [36] M. Gr, M. Voskoglou, A.-B. M.Salem, and A.-B. Salem. Analogy-based and case-based reasoning: Two sides of the same coin. *International Journal of Applications of Fuzzy Sets and Artificial Intelligence*, 4:5–51, 2014.
- [37] D. Han, C. Zhang, X. Fan, A. Hindle, K. Wong, and E. Stroulia. Understanding android fragmentation with topic analysis of vendor-specific bugs. In *Proceedings of the 2012 19th Working Conference on Reverse Engineering, WCRE '12*, pages 83–92, 2012.
- [38] Hericko, Marjan and Ivkovic, Ales. The size and effort estimates in iterative development. *Inf. Softw. Technol.*, 50(7-8):772–781, 2008.
- [39] A. Idri, A. Abran, and T. M. Khoshgoftaar. *Fuzzy Case-Based Reasoning Models for Software Cost Estimation*, pages 64–96. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.
- [40] A. Idri, F. azzahra Amazal, and A. Abran. Analogy-based software development effort estimation: A systematic mapping and review. *Information and Software Technology*, 58(Supplement C):206 – 230, 2015.
- [41] IEEE. Ieee standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, pages 1–84, 1990.
- [42] M.-A. Jashki, R. Zafarani, and E. Bagheri. Towards a more efficient static software change impact analysis method. In *Proceedings of the 8th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering, PASTE '08*, pages 84–90, 2008.
- [43] S. J. Kabeer, M. Nayebi, G. Ruhe, C. Carlson, and F. Chew. Predicting the Vector Impact of Change - An Industrial Case Study at Brightsquid. In *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, ESEM '17, pages 131–140, 2017.

- [44] H. Kagdi, M. L. Collard, and J. I. Maletic. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *Journal of Software Maintenance and Evolution: Research and Practice*, 19(2):77–131, 2007.
- [45] H. Kagdi, M. Gethers, D. Poshyvanyk, and M. L. Collard. Blending conceptual and evolutionary couplings to support change impact analysis in source code. In *2010 17th Working Conference on Reverse Engineering*, pages 119–128, 2010.
- [46] M. R. Karim, S. M. D. Al Alam, S. J. Kabeer, G. Ruhe, B. Baluta, and S. Mahmud. Applying data analytics towards optimized issue management: An industrial case study. In *Proceedings of the 4th International Workshop on Conducting Empirical Studies in Industry, CESI '16*, pages 7–13, 2016.
- [47] M. R. Karim, G. Ruhe, M. M. Rahman, V. Garousi, and T. Zimmermann. An empirical investigation of single-objective and multiobjective evolutionary algorithms for developer’s assignment to bugs. *Journal of Software: Evolution and Process*, 28(12):1025–1060, 2016.
- [48] M. J. Khan. Applications of case-based reasoning in software engineering: a systematic mapping study. *IET Software*, 8(6):258–268, 2014.
- [49] T. M. Khoshgoftaar, K. Ganesan, E. B. Allen, F. D. Ross, R. Munikoti, N. Goel, and A. Nandi. Predicting fault-prone modules with case-based reasoning. In *Proceedings The Eighth International Symposium on Software Reliability Engineering*, pages 27–35, 1997.
- [50] T. M. Khoshgoftaar and N. Seliya. Comparative assessment of software quality classification techniques: An empirical case study. *Empirical Software Engineering*, 9(3):229–257, 2004.
- [51] T. M. Khoshgoftaar, N. Seliya, and N. Sundaresh. An empirical study of predicting software faults with case-based reasoning. *Software Quality Journal*, 14(2):85–111, 2006.
- [52] R. Kikas, M. Dumas, and D. Pfahl. Using dynamic and contextual features to predict issue lifetime in GitHub projects. In *Proceedings of the 13th International Workshop on Mining*

*Software Repositories*, MSR '16, pages 291–302, 2016.

- [53] R. Lapeña, J. Font, F. Pérez, and C. Cetina. Improving feature location by transforming the query from natural language into requirements. *Proceedings of the 20th International Systems and Software Product Line Conference on - SPLC '16*, pages 362–369, 2016.
- [54] P. A. Laplante and C. J. Neill. The demise of the waterfall model is imminent. *Queue*, 1(10):10–15, 2004.
- [55] C. Larman and V. R. Basili. Iterative and incremental development: A brief history. *Computer*, 36(6):47–56, 2003.
- [56] S. Lehnert. A taxonomy for software change impact analysis. In *Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th Annual ERCIM Workshop on Software Evolution*, IWPSE-EVOL '11, pages 41–50, 2011.
- [57] B. Li, X. Sun, H. Leung, and S. Zhang. A survey of code-based change impact analysis techniques. *Software Testing, Verification and Reliability*, 23(8):613–646, 2013.
- [58] J. Li, G. Ruhe, A. Al-Emran, and M. M. Richter. A flexible method for software effort estimation by analogy. *Empirical Software Engineering*, 12(1):65–106, 2007.
- [59] S. L. Lim and A. Finkelstein. *Anticipating Change in Requirements Engineering*, pages 17–34. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.
- [60] A. D. Lucia, F. Fasano, and R. Oliveto. Traceability management for impact analysis. In *Frontiers of Software Maintenance, 2008. FoSM 2008.*, pages 21–30, 2008.
- [61] A. D. Lucia, F. Fasano, R. Oliveto, and G. Tortora. Recovering traceability links in software artifact management systems using information retrieval methods. *ACM Trans. Softw. Eng. Methodol.*, 16(4), 2007.

- [62] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.
- [63] N. Mansour and N. Baba. Ripple effect in web applications. *IJITWE*, 5(2):1–15, 2010.
- [64] L. Marks, Y. Zou, and A. E. Hassan. Studying the fix-time for bugs in large open source projects. In *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, Promise '11, pages 11:1–11:8, 2011.
- [65] A. K. McCallum. Mallet: A machine learning for language toolkit. <http://mallet.cs.umass.edu>, 2002.
- [66] S. McGee and D. Greer. Software requirements change taxonomy: Evaluation by case study. In *Proceedings of the 2011 IEEE 19th International Requirements Engineering Conference*, RE '11, pages 25–34, 2011.
- [67] E. Mendes, N. Mosley, and S. Counsell. The application of case-based reasoning to early web project cost estimation. In *Proceedings 26th Annual International Computer Software and Applications*, pages 393–398, 2002.
- [68] J. G. Morris and C. M. Mitchell. A case-based support system to facilitate software reuse. In *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, volume 1, pages 232–237, 1997.
- [69] M. Nayebi, S. J. Kabeer, G. Ruhe, C. Carlson, and F. Chew. Hybrid Labels are the New Measure! *IEEE Software*, 35(1):54–57, 2017.
- [70] D. E. Perry, S. E. Sim, and S. Easterbrook. Case studies for software engineers. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 1045–1046, 2006.



- [71] K. Petersen, C. Gencel, N. Asghari, D. Baca, and S. Betz. Action research as a model for industry-academia collaboration in the software engineering context. In *Proceedings of the 2014 International Workshop on Long-term Industrial Collaboration on Software Engineering*, WISE '14, pages 55–62, 2014.
- [72] D. Pfahl, S. Karus, and M. Stavnycha. Improving expert prediction of issue resolution time. In *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, EASE '16, pages 42:1–42:6, 2016.
- [73] M. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.
- [74] J. A. Pow-Sang and R. Imbert. *Effort Estimation in Incremental Software Development Projects Using Function Points*, pages 458–465. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.
- [75] U. Raja. All complaints are not created equal: text analysis of open source software defect reports. *Empirical Software Engineering*, 18(1):117–138, 2013.
- [76] A. Rashwan, O. Ormandjieva, and R. Witte. Ontology-based classification of non-functional requirements in software specifications: A new corpus and svm-based classifier. In *Proceedings of the 2013 IEEE 37th Annual Computer Software and Applications Conference*, COMPSAC '13, pages 381–386, 2013.
- [77] X. Ren, F. Shah, F. Tip, B. G. Ryder, and O. Chesley. Chianti: A tool for change impact analysis of Java programs. *Proc. of the OOPSLA*, 2004.
- [78] E. Ries. *How today's entrepreneurs use continuous innovation to create radically successful businesses*. Crown Business, USA, 1st edition, 2011.
- [79] W. Roongkaew and N. Prompoon. Software engineering tools classification based on swelok taxonomy and software profile. In *2013 Second International Conference on Informatics Applications (ICIA)*, pages 122–128, 2013.

- [80] G. Ruhe and M. O. Saliu. The art and science of software release planning. *IEEE Softw.*, 22(6):47–53, 2005.
- [81] P. Runeson and M. Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Softw. Engg.*, 14(2):131–164, 2009.
- [82] P. Runeson, M. Host, A. Rainer, and B. Regnell. *Case Study Research in Software Engineering: Guidelines and Examples*. Wiley Publishing, 1st edition, 2012.
- [83] C. Santana, F. Leoneo, A. Vasconcelos, and C. Gusmão. *Using Function Points in Agile Projects*, pages 176–191. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.
- [84] T. Savage, M. Revelle, and D. Poshyvanyk. FLAT3: feature location and textual tracing tool. *2010 ACM/IEEE 32nd International Conference on Software Engineering*, 2:255–258, 2010.
- [85] G. Scanniello, A. Marcus, and D. Pascale. Link analysis algorithms for static concept location: an empirical assessment. *Empirical Software Engineering*, 20(6):1666–1720, 2014.
- [86] Shahnewaz, S. M. RELREA - An Analytical Approach Supporting Continuous Release Readiness Evaluation. Master’s thesis, University of Calgary, 2014.
- [87] M. Shepperd. *Case-Based Reasoning and Software Engineering*, pages 181–198. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.
- [88] M. Shepperd and G. Kadoda. Comparing software prediction techniques using simulation. *IEEE Transactions on Software Engineering*, 27(11):1014–1022, 2001.
- [89] J. Śliwerski, T. Zimmermann, and A. Zeller. When do changes induce fixes? *SIGSOFT Softw. Eng. Notes*, 30(4):1–5, 2005.
- [90] C. Spearman. *The nature of "intelligence" and the principles of cognition*. Macmillan, Oxford, England, 1923.

- [91] C. Sun, D. Lo, S. C. Khoo, and J. Jiang. Towards more accurate retrieval of duplicate bug reports. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, pages 253–262, 2011.
- [92] X. Sun, B. Li, H. Leung, B. Li, and J. Zhu. Static change impact analysis techniques: A comparative study. *Journal of Systems and Software*, 109:137 – 149, 2015.
- [93] M. Torchiano and F. Ricca. Impact analysis by means of unstructured knowledge in the context of bug repositories. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '10*, pages 47:1–47:4, 2010.
- [94] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller. How long will it take to fix this bug? In *Proceedings of the Fourth International Workshop on Mining Software Repositories, MSR '07*, pages 1–, 2007.
- [95] B. J. Williams, J. Carver, and R. B. Vaughn. Change risk assessment: Understanding risks involved in changing software requirements. In *Software Engineering Research and Practice*, pages 966–971, 2006.
- [96] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.
- [97] M. B. Zanjani, G. Swartzendruber, and H. Kagdi. Impact analysis of change requests on source code based on interaction and commit histories. In *Proceedings of the 11th Working Conference on Mining Software Repositories, MSR 2014*, pages 162–171, 2014.
- [98] Y. Zhang, D. Lo, X. Xia, T. D. B. Le, G. Scanniello, and J. Sun. Inferring links between concerns and methods with multi-abstraction vector space model. *Proceedings - 2016 IEEE International Conference on Software Maintenance and Evolution, ICSME 2016*, pages 110–121, 2017.

- [99] T. Zimmermann, P. Weisgerber, S. Diehl, and A. Zeller. Mining version histories to guide software changes. In *Proceedings of the 26th International Conference on Software Engineering*, ICSE '04, pages 563–572, 2004.