From Bugs to Decision Support – Leveraging Historical Issue Reports in Software Evolution

Markus Borg

Doctoral Thesis, 2015
Department of Computer Science
Lund University

This thesis is submitted to the Research Education Board of the Faculty of Engineering at Lund University, in partial fulfilment of the requirements for the degree of Doctor of Philosophy in Engineering.

LU-CS-DISS: 2015-2
Dissertation 46, 2015

ISBN 978-91-7623-305-4 (printed version)
ISBN 978-91-7623-306-1 (electronic version)
ISSN 1404-1219

Department of Computer Science
Lund University
Box 118
SE-221 00 Lund
Sweden

Email: markus.borg@cs.lth.se
WWW: http://cs.lth.se/markus_borg

Cover art: "Taming the bugs" by Hannah Oredsson
Typeset using LaTeX
Printed in Sweden by Tryckeriet i E-huset, Lund, 2015

ABSTRACT

Software developers in large projects work in complex information landscapes and staying on top of all relevant software artifacts is an acknowledged challenge. As software systems often evolve over many years, a large number of issue reports is typically managed during the lifetime of a system, representing the units of work needed for its improvement, e.g., defects to fix, requested features, or missing documentation. Efficient management of incoming issue reports requires the successful navigation of the information landscape of a project.

In this thesis, we address two tasks involved in issue management: Issue Assignment (IA) and Change Impact Analysis (CIA). IA is the early task of allocating an issue report to a development team, and CIA is the subsequent activity of identifying how source code changes affect the existing software artifacts. While IA is fundamental in all large software projects, CIA is particularly important to safety-critical development.

Our solution approach, grounded on surveys of industry practice as well as scientific literature, is to support navigation by combining information retrieval and machine learning into Recommendation Systems for Software Engineering (RSSE). While the sheer number of incoming issue reports might challenge the overview of a human developer, our techniques instead benefit from the availability of ever-growing training data. We leverage the volume of issue reports to develop accurate decision support for software evolution.

We evaluate our proposals both by deploying an RSSE in two development teams, and by simulation scenarios, i.e., we assess the correctness of the RSSEs' output when replaying the historical inflow of issue reports. In total, more than 60,000 historical issue reports are involved in our studies, originating from the evolution of five proprietary systems for two companies. Our results show that RSSEs for both IA and CIA can help developers navigate large software projects, in terms of locating development teams and software artifacts. Finally, we discuss how to support the transfer of our results to industry, focusing on addressing the context dependency of our tool support by systematically tuning parameters to a specific operational setting.

POPULAR SUMMARY

Ta dig fram i informationslandskapet med buggar som hävstång

Markus Borg, *Inst. för Datavetenskap, Lunds Universitet*

Utvecklare i stora mjukvaruprojekt måste orientera sig i enorma informationslandskap. Brist på överblick innebär ett hårt slag mot produktiviteten. Mönster ifrån tidigare buggar kan guida framtida underhållsarbete.

– Åhh, jag VET att den finns här! Tilda måste hitta den där gränssnittsbeskrivningen för att komma vidare. Hon är stressad eftersom hon lovat slutföra förändringarna i källkoden i eftermiddag. Hon sliter sitt hår medan hon frenetiskt letar bland dokumenten.

Som vanligt tvingas Tilda bläddra runt längre än hon vill i dokumenthanteringssystemets svårbegripliga struktur. Hon har bara letat i en kvart den här gången, men det känns som att hon har lagt en timme på något som borde gå på nolltid. Vad hon inte heller vet är att en kollega i Tyskland letade efter precis samma kravspecifikation för bara två veckor sedan.

Situationen är vanlig i stora mjukvaruutvecklingsprojekt. Systemen växer sig allt mer komplexa och underhålls under allt längre tid. Både källkod och relaterad dokumentation är så omfattande att enskilda utvecklare inte kan överblicka informationslandskapet.

Rätt information i rättan tid

Ett stort mjukvaruprojekt utgör en besvärlig informationsrymd att navigera. Det finns tiotusentals dokument som representerar olika specifikationer, beskrivningar, källkodsfiler och testskript. Informationen är utspridd över olika databaser som sällan har bra sökverktyg. Dessutom förändras informationen konstant i takt med att systemet utvecklas.

Mjukvaruutveckling är ett kunskapsintensivt arbete. Att snabbt hitta rätt information är kritiskt för produktiviteten. Studier visar att utvecklare lägger 20-30% av sin arbetstid på att leta information. Tillstånd av sk. "information overload" är vanligt – det finns mer information tillgänglig än vad man kan hantera.

Digitala spår och datorträning

Varje gång en utvecklare rättar en bugg lämnas digitala spår i informationslandskapet. Förändringar i källkod och dokumentation lagras med tidsangivelser. Ju mer projekt-

historik som finns tillgänglig, desto tydligare mönster kan man hitta.

Maskininlärning är ett samlingsnamn för tekniker som låter en dator på egen hand finna mönster i data. Processen kallas träning. En tränad dator kan användas för att prediktera vad som kommer att hända härnäst, eller för att sortera nya data baserat på vad som tidigare bearbetats.

Forskare har föreslagit att låta en dator tränas på hur man tidigare har hanterat buggar i ett projekt. Då kan datorn hitta mönster både i själva inflödet av buggar och i de digitala spår som utvecklarna lämnar efter sig när de gör sina rättningar. Kanske brukar Tilda rätta buggar som handlar om minneshantering? Eller är det så att när minnesbuggar rättas ändras ofta en viss specifikation?

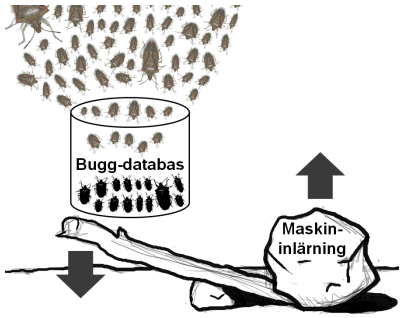
Vägvisare i landskapet

I vårt arbete har vi utvecklat rekommendationssystem baserade på maskininlärning. Inspirerat av hur kundanpassade köprekommendationer presenteras inom e-handel presenterar vårt system information som bedöms vara relevant för utvecklarens pågående arbetsuppgift. Vårt system följer de digitala spår som utvecklare lämnar efter sig och presenterar de mest upptrampade stigarna i ett tydligt gränssnitt.

Vi använder även maskininlärning för att föreslå vilken utvecklare som är mest lämpad att undersöka en ny bugg. På så vis kan en utvecklingsorganisation minska det manuella fördelningsarbetet. Förhoppningen är även att fler buggar hamnar rätt direkt – en minskning av "heta potatisar" som kastas runt bland utvecklarna!

Tack vare maskininlärning låter vi mängden historiska buggar verka till vår fördel. Människor har svårt att överblicka stora mängder buggar. En dator däremot hittar tydligare mönster ju mer historisk data som finns tillgänglig. Fler buggar leder till säkrare rekommendationer – antalet buggar ger våra verktyg en hävstångseffekt.

Vi har utvärderat vårt verktyg på data från industriella utvecklingsprojekt. Totalt har vi studerat fler än 60.000 historiska buggar från fem olika projekt inom säkerhetskritiska automationssystem och grundläggande IT-infrastruktur. Genom att spela upp det historiska inflödet av buggar visar vi att vårt verktyg presenterar rekommendationer som är lika bra som de mänskliga besluten – men det går blixtsnabbt! En människa behöver en stund för att göra sin analys, men vårt verktyg levererar förslag på bråkdelen av en sekund.



Slutsats

Vår forskning visar att en dator tränad på de digitala spår som automatiskt lagras under ett mjukvaruprojekt kan hjälpa utvecklare att hitta rätt. De inblandade utvecklarna trampar kollektivt upp stigar i informationslandskapet. Dessa stigar utgör en värdefull resurs.

Rekommendationer som baseras på maskininlärning blir skarpare ju mer träningsdata som finns tillgänglig. Med våra idéer satta i drift skulle nyrekryterade utvecklare enkelt kunna ta del av den erfarenhet som vuxit fram efter år av utvecklingsarbete – man skulle få till en automatisk kunskapsöverföring.

Och Tilda då? Jo, hon skulle ha blivit rekommenderad det där dokumentet hon letade efter. Tysken hade ju trampat upp spår bara ett par veckor tidigare...

SUMMARY IN 140 CHARACTERS

"Humans obscured by bug overload, but machine learning benefits from plentiful training data. Practitioners confirm value of developed tools."

- *Tiny Transactions on Computer Science* (@TinyToCS), Volume 3, 2015

ACKNOWLEDGEMENTS

This work was funded by the Industrial Excellence Center EASE – Embedded Applications Software Engineering.

Completing a Ph.D. is the result of a long journey, and many people deserve recognition. My first computer experience was gained through a Spectravideo SV-328 equipped with a 3.6 MHz CPU and 128 KB RAM[1]. Thanks to my older brothers, I was exposed to source code from the start. My friend Magnus and I spent hours looking at BASIC program listings, and we did our best to copy bits and pieces for our own creations. Back then, the main software quality attribute I cared about was lines of code; nothing was more rewarding than writing long programs to make the tape counter show a lot of supply reel revolutions.

Twenty years later I had the opportunity to join the Software Engineering Research Group in Lund as a Ph.D. student, and most importantly, I want to extend my deepest gratitude to my supervisor Professor Per Runeson. You have profoundly changed my perspective on software, so that it now stretches far beyond the quality characteristics measurable by a tape counter. Thank you for guiding me into the world of research. Thank you also for your support in the personal sphere, both our time in Raleigh and the hatted wedding drive will always be remembered.

An array of thanks goes to the rest of my colleagues in the research group, and the CS department at Lund University. Particularly, Professor Björn Regnell for being an inspirational co-supervisor, Professor Martin Höst for our countless lunches together, and Dr. Krzysztof Wnuk for being a good friend. Special thanks also go to my co-authors within the EASE project; especially Dr. Elizabeth Bjarnason, Dr. Emelie Engström, and Michael Unterkalmsteiner.

During the past few years, I have learned vital research tools from several collaborators. Although I have learned from all my 33 co-authors listed in this thesis, some of you deserve special mention. Leif Jonsson: thanks for all interesting group calls over Skype; MLAFL Bayes was surprisingly vetoed, but your curves have had a significant impact in other ways. Finding you at ICSE 2013

[1]Technical specification: CPU Zilog Z80A @ 3.6 MHz, 64 KB RAM + 64 KB RAM expansion cartridge, VDC TI TMS9918 with 16 KB VRAM (resolution 256x192, 16 colors), sound chip GI AY-3-8910, peripheral data cassette recorder.

was a happy turn of events on this journey. Dr. Dietmar Pfahl: thanks for all your support to me as a research first-timer during our discussions. Many of the fundamental skills needed for scientific research and academic writing were first learned from you and your contribution is much appreciated.

I have also had the privilege to work with excellent industry partners. Above all, I want to express my gratitude to my ABB colleagues from three different continents. Special thanks go to Dr. Magnus Larsson, Will Snipes, Ganesh Rao, Andreas Ekstrand, Artour Klevin, Håkan Gustavsson, and Łukasz Serafin. Your support was essential and will never be forgotten. As an empirical researcher, many people have provided me with valuable data. Thanks to everyone involved, whether as survey respondents, experimental subjects, or case study participants. Without you, there would be no thesis.

Finally, I want to express my deepest thanks to my family. In this most important area resides a small apology for my tendency to bring out the laptop whenever the opportunity arises; I am aware that my anti-idling tendencies may sometimes also be anti-social. The highest price of this thesis has been paid by the people closest to me, since a significant part of the text has been written outside office hours. Sorry for sacrificing quality time at home, in Rydebäck and Gräsås, in cars, buses, trains, parks, on beaches, and the TV couch. Thanks also go to my parents Lena and Örjan for supporting me through the entire education system, and my brothers Magnus and Pontus for computer sciencepiration.

Saving the best for last, Marie and Tilda, thank you both for all the love and support, and for letting me spend time with my rackets when I needed it. Marie: thanks for all the reviews and even the typesetting of two of the papers. Tilda: your arrival surely pushed my creativity to the limits, as I needed to simultaneously come up with two distinctive identifiers: a name for you and a title for the thesis. You are the best reality check; being with you is such a simple way to find out what really matters. Thank you. I have passed considerable milestones in recent months on rather different levels in life. Building on them in the next chapter will unquestionably be at least as exciting as the journey has been so far.

Press play on tape
Markus Borg
Malmö, April 2015

LIST OF PUBLICATIONS

In the introduction chapter of this thesis, the included and related publications listed below are referred to by Roman numerals. To distinguish the included publications in the running text, a preceding 'Paper' is added.

Publications included in the thesis

I **Challenges and Practices in Aligning Requirements with Verification and Validation: A Case Study of Six Companies**
Elizabeth Bjarnason, Per Runeson, Markus Borg, Michael Unterkalmsteiner, Emelie Engström, Björn Regnell, Giedre Sabaliauskaite, Annabella Loconsole, Tony Gorschek, and Robert Feldt
Empirical Software Engineering, 19(6), pages 1809-1855, 2014.

II **Recovering from a Decade: A Systematic Literature Review of Information Retrieval Approaches to Software Traceability**
Markus Borg, Per Runeson, and Anders Ardö
Empirical Software Engineering, 19(6), pages 1565-1616, 2014.

III **Automated Bug Assignment: Ensemble-based Machine Learning in Large Scale Industrial Contexts**
Leif Jonsson, Markus Borg, David Broman, Kristian Sandahl, Sigrid Eldh, and Per Runeson
Under revision in *Empirical Software Engineering*, 2015.

IV **Supporting Change Impact Analysis Using a Recommendation System: An Industrial Case Study in a Safety-Critical Context**
Markus Borg, Krzysztof Wnuk, Björn Regnell, and Per Runeson
To be submitted, 2015.

V **TuneR: A Framework for Tuning Software Engineering Tools with Hands-On Instructions in R**
Markus Borg
Submitted to a journal, 2015.

Related publications

VI **Do Better IR Tools Improve the Accuracy of Engineers' Traceability Recovery?**
Markus Borg, and Dietmar Pfahl
In *Proc. of the International Workshop on Machine Learning Technologies in Software Engineering* (MALETS'11), pages 27-34, Lawrence, Kansas, United States, 2011.

VII **Towards Scalable Information Modeling of Requirements Architectures**
Krzysztof Wnuk, Markus Borg, and Saïd Assar
In *Proc. of the 1st International Workshop on Modelling for Data-Intensive Computing* (MoDIC'12), pages 141-150, Florence, Italy, 2012.

VIII **Findability through Traceability - A Realistic Application of Candidate Trace Links?**
Markus Borg
In *Proc. of the 7th International Conference on Evaluation of Novel Approaches to Software Engineering* (ENASE'12), pages 173-181, Wrocław, Poland, 2012.

IX **Industrial Comparability of Student Artifacts in Traceability Recovery Research - An Exploratory Survey**
Markus Borg, Krzysztof Wnuk, and Dietmar Pfahl
In *Proc. of the 16th European Conference on Software Maintenance and Reengineering* (CSMR'12), pages 181-190, Szeged, Hungary, 2012.

X **Evaluation of Traceability Recovery in Context: A Taxonomy for Information Retrieval Tools**
Markus Borg, Per Runeson, and Lina Brodén
In *Proc. of the 16th International Conference on Evaluation & Assessment in Software Engineering* (EASE'12), pages 111-120, Ciudad Real, Spain, 2012.

XI **Advancing Trace Recovery Evaluation - Applied Information Retrieval in a Software Engineering Context**
Markus Borg
Licentiate Thesis, Lund University, Sweden, 2012.

XII **Confounding Factors When Conducting Industrial Replications in Requirements Engineering**
David Callele, Krzysztof Wnuk, and Markus Borg
In *Proc. of the 1st International Workshop on Conducting Empirical Studies in Industry* (CESI'13), pages 55-58, San Francisco, California, United States, 2013.

XIII **Enabling Traceability Reuse for Impact Analyses: A Feasibility Study in a Safety Context**
Markus Borg, Orlena Gotel, and Krzysztof Wnuk
In *Proc. of the 7th International Workshop on Traceability in Emerging Forms of Software Engineering* (TEFSE'13), pages 72-78, San Francisco, California, United States, 2013.

XIV **Analyzing Networks of Issue Reports**
Markus Borg, Dietmar Pfahl, and Per Runeson
In *Proc. of the 17th European Conference on Software Maintenance and Reengineering* (CSMR'13), pages 79-88, Genova, Italy, 2013.

XV **IR in Software Traceability: From a Bird's Eye View**
Markus Borg, and Per Runeson
In *Proc. of the 7th International Symposium on Empirical Software Engineering and Measurement* (ESEM'13), pages 243-246, Baltimore, Maryland, United States, 2013.

XVI **Changes, Evolution, and Bugs - Recommendation Systems for Issue Management**
Markus Borg, and Per Runeson
In **Recommendations Systems in Software Engineering**, *Martin Robillard, Walid Maalej, Robert Walker, and Thomas Zimmermann* (Eds.), pages 477-509, Springer, 2014.

XVII **Supporting Regression Test Scoping with Visual Analytics**
Emelie Engström, Mika Mäntylä, Per Runeson, and Markus Borg
In *Proc. of the 7th International Conference on Software Testing, Verification and Validation* (ICST'14), pages 283-292, Cleveland, Ohio, United Stats, 2014.

XVIII **Development of Safety-Critical Software Systems - A Systematic Map**
Sardar Muhammad Sulaman, Alma Orucevic-Alagic, Markus Borg, Krzysztof Wnuk, Martin Höst, and Jose-Luis de la Vara
In *Proc. of the Euromicro Conference on Software Engineering and Advanced Applications* (SEAA'14), pages 17-24, Verona, Italy, 2014.

XIX **Revisiting the Challenges in Aligning RE and V&V: Experiences from the Public Sector**
Jacob Larsson, and Markus Borg
In *Proc. of the 1st International Workshop on Requirements Engineering and Testing* (RET'14), pages 4-11, Karlskrona, Sweden, 2014.

XX **Workshop Summary of the 1st International Workshop on Requirements and Testing (RET'14)**
Michael Felderer, Elizabeth Bjarnason, Michael Unterkalmsteiner, Mirko Morandini, and Matthew Staats
Technical Report, arXiv:1410.3401, 2014.

XXI **A Replicated Study on Duplicate Detection: Using Apache Lucene to Search among Android Defects**
Markus Borg, Per Runeson, Jens Johansson, and Mika Mäntylä
In *Proc. of the 8th International Symposium on Empirical Software Engineering and Measurement* (ESEM'14), Torino, Italy, 2014.

XXII **Survey on Safety Evidence Change Impact Analysis in Practice: Detailed Description and Analysis**
Jose-Luis de la Vara, Markus Borg, Krzysztof Wnuk, and Leon Moonen
Technical Report, Simula, 2014.

XXIII **Navigating Information Overload Caused by Automated Testing: A Clustering Approach in Multi-Branch Development**
Nicklas Erman, Vanja Tufvesson, Markus Borg, Anders Ardö, and Per Runeson
In *Proc. of the 8th International Conference on Software Testing, Verification and Validation* (ICST'15), Graz, Austria, 2015.

XXIV **An Industrial Case Study on the Use of Test Cases as Requirements**
Elizabeth Bjarnasson, Michael Unterkalmsteiner, Emelie Engström, and Markus Borg
To appear in *Proc. of the 16th International Conference on Agile Software Development* (XP'15), Helsinki, Finland, 2015.

XXV **The More the Merrier: Leveraging on the Bug Inflow to Guide Software Maintenance**
Markus Borg, and Leif Jonsson
Tiny Transactions on Computer Science, Volume 3, 2015.

XXVI **Using Text Clustering to Predict Defect Resolution Time: A Conceptual Replication and an Attempted Proof-of-Concept**
Saïd Assar, Markus Borg, and Dietmar Pfahl
Under revision in *Empirical Software Engineering*, 2015.

Contribution statement

Collaboration is central in research. All papers included in this thesis, except Paper V, have been co-authored with other researchers. The authors' individual contributions to Papers I-IV are as follows:

Paper I
The first paper included in the thesis comprises a large research effort with many people involved. The most senior researchers defined the overall goals of the study: Prof. Björn Regnell, Prof. Tony Gorschek, Prof. Per Runeson, and Prof. Robert Feldt. Dr. Annabella Loconsole, Dr. Giedre Sabaliauskaite, and Dr. Emelie Engström designed and planned the study. All ten authors were involved in the data collection, i.e., conducting interviews with practitioners. Dr. Elizabeth Bjarnason, Markus Borg, Dr. Emelie Engström, and Michael Unterkalmsteiner did most of the data analysis, comprising large quantities of qualitative data. Finally, Dr. Elizabeth Bjarnason and Prof. Per Runeson performed most of the writing. All authors then reviewed the paper prior to publication.

Paper II
The literature review reported in Paper II was conducted in parallel to the study in Paper I. Markus Borg was the first author with the main responsibility for the research effort. The study was co-designed with Prof. Per Runeson. Markus Borg did most of the data analysis, which was then validated by Prof. Per Runson and Dr. Anders Ardö. Markus Borg wrote a majority of the text, and the co-authors contributed with constructive reviews.

Paper III
Six authors contributed to Paper III, describing a controlled experiment with industry partners. Leif Jonsson proposed using stacked generalization for issue assignment, and developed the tool that is evaluated in the experiment. The study was designed by Leif Jonsson, Markus Borg, Dr. David Broman, and Prof. Kristian Sandahl. Leif Jonsson and Markus Borg collected data from two separate companies, and together analyzed the results. The results were than reviewed by the other four authors. Markus Borg organized the writing process, and wrote most of the text.

Paper IV
The case study reported in Paper IV was co-authored by four authors. Markus Borg developed the tool that is evaluated in the study. Markus Borg also designed the study, collected and analyzed all data, and did most of the writing. Dr. Krzysztof Wnuk, Prof. Björn Regnell, and Prof. Per Runeson provided feedback during the study and reviewed the paper.

CONTENTS

Introduction **1**
1 Background . . . 3
2 Related Work . . . 9
3 Research Overview . . . 13
4 Research Methodology . . . 15
5 Results . . . 23
6 Synthesis . . . 30
7 Threats to Validity . . . 36
8 Conclusion and Future Work . . . 39

Part I: The Exploratory Phase **43**

I Challenges and Practices in Aligning Requirements with Verification and Validation: A Case Study of Six Companies **45**
1 Introduction . . . 46
2 Related Work . . . 47
3 Case Study Design . . . 50
4 Results . . . 62
5 Discussion . . . 88
6 Conclusions . . . 93

II Recovering from a Decade: A Systematic Review of Information Retrieval Approaches to Software Traceability **95**
1 Introduction . . . 96
2 Background . . . 97
3 Related Work . . . 102
4 Method . . . 111
5 Results . . . 119
6 Discussion . . . 127
7 Summary and Future Work . . . 133

Part II: The Solution Phase **141**

III Automated Bug Assignment: Ensemble-based Machine Learning in Large Scale Industrial Contexts **143**
1 Introduction . . . 144
2 Machine Learning . . . 146
3 Related Work on Automated Bug Assignment . . . 148
4 Case Descriptions . . . 156
5 Method . . . 159
6 Results and Analysis . . . 173
7 Threats to Validity . . . 180
8 Discussion . . . 184
9 Conclusions and Future Work . . . 188

IV Supporting Change Impact Analysis Using a Recommendation System: An Industrial Case Study in a Safety-Critical Context **191**
1 Introduction . . . 192
2 Background and Related Work . . . 194
3 Industrial Context Description . . . 201
4 Approach and ImpRec . . . 204
5 Research Method . . . 209
6 Results and Interpretation . . . 216
7 Threats to Validity . . . 228
8 Discussion . . . 231
9 Conclusion and Future Work . . . 237

Part III: The Utilization Phase **241**

V TuneR: A Framework for Tuning Software Engineering Tools with Hands-On Instructions in R **243**
1 Introduction . . . 244
2 Background . . . 246
3 Related Work on Parameter Tuning in Software Engineering . . . 251
4 ImpRec: An RSSE for Automated Change Impact Analysis . . . 253
5 TuneR: An Experiment Framework and a Hands-on Example . . . 255
6 Tuning ImpRec Using Exhaustive Search . . . 279
7 Discussion . . . 281
8 Conclusion . . . 284

Bibliography **287**
References . . . 290

INTRODUCTION

The information landscape of a large software engineering project is complex. First, the *sheer volume of information* that developers maintain in large projects threatens the overview, as tens of thousands of development artifacts are often involved [177, 391]. Second, developers work collaboratively on *heterogeneous development artifacts* stored in various software repositories such as source code repositories, requirements databases, test management systems, and general document management systems. Often the databases have *poor interoperability* [173], thus they turn into "*information silos*", i.e., simple data storage units with little transparency for other tools. Third, as source code is easy to modify, at least when compared to the accompanied hardware, the software system under development continuously evolves during a project. Not only does the source code evolve, but the related development artifacts should also co-evolve to reflect the changes, e.g., design documents and test case descriptions might require continuous updates [111]. Consequently, staying on top of the information landscape in large software engineering projects constitutes a significant challenge for both developers and managers [403].

In knowledge-intensive work such as software engineering projects, quick and concise access to information is fundamental. If the project environment does not provide sufficient support for navigation and retrieval, considerable effort is wasted on locating the relevant information [263]. Unfortunately, large software engineering projects are threatened by information overload, i.e., "*a state where individuals do not have the time or capacity to process all available information*" [172]. Freund *et al.* reported that software engineers spend about 20-30% of their time consulting various software repositories, but still often fail to fulfil their information needs [189]. Dagenais *et al.* showed that poor search functionality in information repositories constitutes an obstacle for newcomers entering new software projects [132]. This thesis reinforces the importance of information access in software engineering, by reporting that the sheer volume of information threatens the alignment between requirements engineering and testing [Paper I].

Issue trackers constitute examples of software repositories containing large amounts of information. As there is typically a continuous inflow of issue reports, individual developers often struggle to sustain an overview of the current

state of the issue tracker [22]. The inflow in large projects makes activities such as duplicate management, prioritization, and work allocation time-consuming and inefficient [XVI] [53, 257]. On the other hand, previous works argue that the issue repository is a *key collaborative hub* in software engineering projects, and that it can be harnessed to provide decision support to developers. Čubranić *et al.* developed Hipikat, a recommendation system to help newcomers in open source communities navigate existing information, by mining a "project memory" from the issue repository [127]. Anvik and Murphy presented automated decision support for several activities involved in issue management, all based on information stored in issue repositories [24].

In this thesis, we also view the issue repository as a key collaborative hub, but we increase the granularity further by considering each individual issue report as an important juncture in the software engineering information landscape. We have previously shown that issue reports can connect software artifacts that are stored in separate databases [XIII], i.e., issue reports are a way to break information silos. In software engineering contexts where the change management process is rigid, every corrective change committed as part of an issue resolution must be documented [XXII]. As such, the trace links from issue reports to artifacts in various repositories, e.g., requirements and test case descriptions, turn into trails in the information landscape, created by past engineers as part of their maintenance work.

We apply techniques from Machine Learning (ML), Information Retrieval (IR), and Recommendation Systems for Software Engineering (RSSE) to detect patterns in the historical issue reports, predict relationships, and provide developers with actionable decision support. As the techniques we rely on generally perform better the more data that are available [37, 161], we leverage the daunting inflow of issue reports to assist navigation in the software engineering landscape. We focus on two specific tasks involved in issue management: 1) *issue assignment*, i.e., the initial task of deciding who should investigate an issue report, and 2) *change impact analysis*, i.e., a subsequent task of investigating how a proposed change to the software will affect the rest of the system. In contrast to previous work, we study *issue assignment at team level* rather than for individual developers, and we focus on *change impact analysis of non-code artifacts*, i.e., development artifacts that are not source code.

We develop tools to support issue management and report evaluations conducted in two companies. While most previous work on tool support for issue management focuses on open source software development projects, instead we target proprietary projects. We evaluate our tools using experiments *in silico*, and our proposal to support change impact analysis using an RSSE is also studied *in situ*, i.e., we deploy our RSSE in industry and observe how it performs with users in a real setting. Thus, the user-oriented research we present in this thesis answers calls from both traceability researchers [202], and the RSSE community [402], regarding the need for industrial case studies. Finally, as our studies indicate that

the performance of our tools is highly context-dependent, we present guidelines on how to tune tools for a specific operational context. To summarize, the main contributions of this thesis are:

- Two rich surveys of challenges and solutions related to information access in large projects, covering both state-of-practice and state-of-the-art.
- A comprehensive evaluation of automated issue assignment in two proprietary contexts.
- An in-depth case study on automated change impact analysis, involving developers in the field.
- A discussion on context and data dependencies, with hands-on guidelines for parameter tuning using state-of-the-art experimental designs.

1 Background

This section contains a brief overview of some software engineering concepts that are central to this thesis, and introduce the principal techniques and approaches involved in our solution proposals. All sections are condensed, mainly focusing on establishing the terminology used throughout the thesis, with pointers to relevant background sections in the included papers.

1.1 Issue Management

Issue management is a fundamental activity in software maintenance and evolution, comprising reporting, assignment, tracking, resolution, and archiving of issue reports [51]. The issue management process in an organization is tightly connected to the underlying information system, i.e., the *issue tracker*. The issue tracker is not only an archival database, but is also an important facilitator for communication and coordination in a software project. Frequently used issue trackers in industry include: Bugzilla, JIRA, and Trac [100]. The various activities involved in issue management are generally known to be a costly part of the lifecycle of a software-intensive system [22, 125, 338].

An *issue report* contains information about observed misbehavior regarding a software system. Issue reports are typically structured according to the input form of the issue tracker, combining drop down menus and free-text fields. According to Herzig and Zeller [218], the standard components that describe an issue report are:

- *Environment information*, e.g., product, version, and operating system, helping developers to isolate an issue.

- A *free-text description*, written by the submitter, presenting the observed issue and, hopefully, steps to reproduce the issue.
- *Management fields*, e.g., issue type, assignee, and priority, used internally by developers and managers to organize work.

As a development organization processes an issue report, it moves through a series of states. Figure 1 presents an overview of the principal workflow we consider in this thesis, i.e., issue management in a large development organization in which a *Change Control Board* (CCB) decides how to act on individual issue reports. When an issue report is submitted, it starts in the *New* state. The CCB then assigns the issue report to the appropriate development team, i.e., it enters the *Assigned* state. The developer tries to replicate the issue and reports his experiences, and the issue report moves to the *Triaged* state. Based on the triage, the CCB either moves the issue report to the *Accepted* or the *Rejected* state. If the issue report is accepted, the developer gets it back and starts designing a resolution. When the developer has proposed a resolution, the issue report enters the *Change Analyzed* state. The CCB then decides if the proposed changes to the system are acceptable, and moves the issue report to either the *Change Accepted* state or the *Rejected* state. If the change is accepted, the developer implements the change and moves the issue report to the *Resolved* state. When the change has been committed, a developer or tester verifies that the issue has been properly addressed, and the issue report moves to the *Verified* state. Finally, once everything is complete, CCB moves the issue report to the *Closed* state. We further discuss issue management in Papers III and IV, focusing on issue assignment and change impact analysis, respectively.

Cavalcanti *et al.* recently published a systematic mapping study on challenges and opportunities for issue trackers [100]. They analyzed 142 studies on the topic, and also compared their findings with the features offered by state-of-practice issue trackers. Several challenges of issue assignment have been addressed by previous research. According to Cavalacanti *et al.*, the most commonly targeted challenges are: 1) *issue assignment* (28 studies), 2) *duplicate detection* (20 studies), 3) *resolution time prediction* (18 studies), 4) *quality assessment* (15 studies), and 5) *change impact analysis* (14 studies). Among the top-5 challenges, this thesis contributes to the issue assignment [Paper III] and change impact analysis [Paper IV]. However, we have also studied duplicate detection [XXI] and resolution time prediction [XXVI] in parallel work. Finally, Cavalcanti *et al.* report that few ideas from research on assisted issue management have been adopted in state-of-practice issue trackers, and conclude that more empirical studies are needed on how the proposed tool support can support software engineering. Another recent literature review by Zhang *et al.* confirms that state-of-practice issue trackers provide little automated support for issue management [493]. Also, Zhang *et al.* hypothesize that the main obstacle for dissemination of research results to industry is that "*none of the previous [automated] approaches has achieved satisfactory accuracy*".

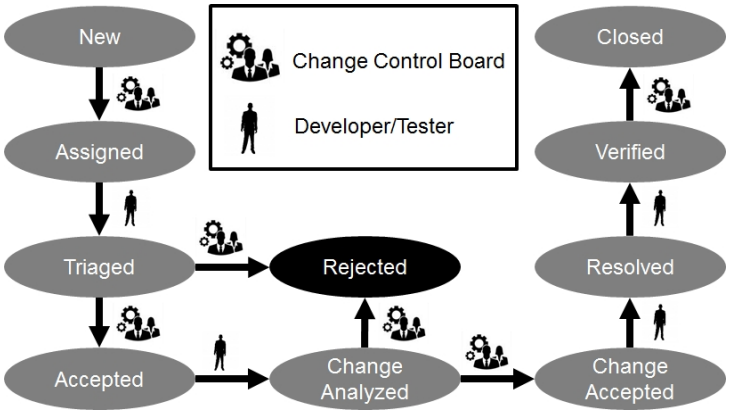


Figure 1: Overview of the issue management workflow in a large software development organization with a Change Control Board (CCB). The arrows depict the primary transitions between the states of an issue report; an issue report can also be changed back to an earlier state.

1.2 Traceability and Change Impact Analysis

Traceability has been discussed in software engineering since the pioneering NATO Working Conference on Software Engineering in 1968. Randall argued that a developed software system should "*contain explicit traces of the design process*" [386]. Boehm mentioned traceability as an important contemporary software engineering challenge in a state-of-the-art survey from 1976, and predicted traceability to become a future research trend [66]. Concurrently, industrial practice acknowledged traceability as a vital part of high-quality software, and by the 1980s several development standards had emerged that mandated traceability maintenance [162].

In the 1990s, the amount of traceability research increased with the advent of requirements engineering as a dedicated research field. Gotel and Finkelstein identified the lack of a common traceability definition, and proposed a definition tailored for the requirements engineering community: "*requirements traceability refers to the ability to describe and follow the life of a requirement, in both forwards and backwards direction*" [204]. According to a systematic literature review by Torkar *et al.*, this definition is the most commonly cited in traceability research [449].

In the 2000s, the growing interest in agile development methods made many organizations downplay traceability. Agile developers often consider traceability management to be a burdensome activity that does not generate return on investment [110]. Still, traceability remains non-negotiable in development of safety-critical systems. Safety standards such as ISO 26262 in the automotive industry [242] and IEC 61511 in the process industry sector [238] explicitly requires

traceability through the development lifecycle.

In 2012, Cleland-Huang *et al.* published the first (edited) book on software and systems traceability [113]. The book summarizes research on traceability, and contains several fundamental definitions. A large part of the traceability research community, organized in CoEST[2], contributed to the book. *Traceability*, in general, is defined as the "*potential for traces to be established and used*", while *requirements traceability* is again defined according to Gotel and Finkelstein's paper from 1994 [204]. Paper II contains more background on traceability in software engineering, as well as an overview of traceability research. Apart from the basic definitions of traceability and requirements traceability, we use the following terminology in this thesis:

- A *trace artifact* is a traceable unit of data.
- A *trace link* denotes an association forged between two trace artifacts, e.g., dependency, refinement or conflict.
- A *trace* is a triplet of two trace artifacts connected by a trace link.
- *Tracing* is the activity of establishing or using traces.
- *Trace capture* is an approach to establish trace links concurrently with the creation of the trace artifacts that they associate.
- *Trace recovery* is an approach to establish trace links after the trace artifacts that they associate have been generated or manipulated.
- A *tracing tool* is any tool that supports tracing.

A concept closely related to traceability is *Change Impact Analysis* (CIA), defined by Bohner as "*identifying the potential consequences of a change, or estimating what needs to be modified to accomplish a change*" [67]. CIA is a cognitive process of incrementally adding items to a set of candidate impact. Several researchers report that CIA for complex software-intensive systems is both a tedious and an error-prone activity [103, 293, 337]. However, analogous to traceability maintenance, CIA is mandated by safety-standards [174, 238, 242]. Access to traces might support developers with CIA [128, 304], a statement that is often brought forward as a rationale for costly traceability efforts within software projects.

Most CIA work in industry is manual [XXII] [469], and research on CIA tools have been highlighted as an important direction for future work [68]. Also, two recent reviews of scientific literature shows that most research on CIA is limited to impact on source code [293, 301]. However, as stated in Lehnert's review: "*more attention should be paid on linking requirements, architectures, and code to enable*

[2]The Center of Excellence for Software Traceability, www.coest.org

comprehensive CIA" [293, pp. 26]. Especially in safety-critical development, it is critical to also analyze how a change to a software system affects artifact types that are not source code, e.g., whether any requirements are affected, or which test cases should be selected for regression testing. In this thesis, we specifically focus on CIA of development artifacts that are not source code, i.e., non-code artifacts. Paper IV contains a more thorough introduction to CIA.

1.3 Techniques and Approaches Applied in the Thesis

The solution approaches presented in the thesis are inspired by research in several fields, but mainly rely on information retrieval, machine learning, and recommendation systems. This subsection lists fundamental definitions, and provides pointers to more extensive background sections in the included papers.

Information Retrieval (IR) is "*finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers)*" [325]. If a retrieved document satisfies such a need, we consider it *relevant*. We solely consider text retrieval in the study, yet we follow convention and refer to it as IR. We define NL and Natural Language Processing (NLP) as follows [306]: "*NL text is written in a language used by humans to communicate to one another*", and "*NLP is a range of computational techniques for analyzing and representing NL text*". Paper II contains a thorough overview of IR in software engineering.

Machine Learning (ML) is an area within computer science concerned with how computer programs can learn and improve at performing a specific task when trained on historical data. ML is divided into *unsupervised learning* and *supervised learning*. Unsupervised learning detects patterns in data, e.g., applicable in clustering and anomaly detection. We have explored clustering of issue reports based on their textual content in related work [XXVI], but the work in this thesis focuses on supervised learning. In supervised learning each data point is associated with a label or a numerical value. Learning is defined as the ability to generalize from historical data to predict the label (i.e., classification) or value (i.e., regression) of new data points. A supervised ML algorithm is trained on a *training set* and evaluated on a (preferably sequestered) *test set* [62]. Paper III contains an introduction to ML in general and classification in particular.

Recommendation systems provide suggestions for items that are of potential interest for a user [179]. The two main techniques to match users and items are *content-based filtering* and *collaborative filtering*. Content-based filtering finds patterns in the content of items that have been consumed or rated by a user, to find new items that are likely to match his or her interests. Collaborative filtering on the other hand instead identifies users that display similar preference patterns, then their ratings are used to infer recommendations of new items for similar users. Many recommendation systems also combine the two techniques in *hybrid systems*. Robillard *et al.* have proposed a dedicated definition of Recommendation

Table 1: Ten levels of automation as defined by Parasuraman *et al.* [367].

Level	Description. The system...
10	...decides everything, acts autonomously, ignoring the human.
9	...informs the human only if it, the system, decides to.
8	...informs the human only if asked.
7	...executes automatically, then necessarily informs the human.
6	...allows the human a restricted time to veto before automatic execution.
5	...executes that suggestion if the human approves.
4	...suggests one alternative.
3	...narrows the selection down to a few.
2	...offers a complete set of decision/action alternatives.
1	...offers no assistance: human must take all decisions and actions.

Systems for Software Engineering (RSSE): "*a software application that provides information items estimated to be valuable for a software engineering task in a given context*" [403].

Robillard *et al.* report that while the definition is broad on purpose, there still are four specific aspects that all must be fulfilled to qualify as an RSSE instead of a regular software engineering tool [402]. First, the goal of an RSSE is to *provide information* that help developers' decision making. Second, an RSSE *estimates the relevance* of information, i.e., they do not extract facts like compilers or regular expression searches. Third, RSSEs can provide both *novelty and surprise* in discovering new information and *familiarity and reinforcement* by supporting confirmation of existing knowledge. Fourth, an RSSE delivers recommendations for a *specific task in a given context*, as opposed to for example general search tools. We further elaborate on RSSEs, in the context of issue management, in our related book chapter [XVI], and in Paper IV.

In this thesis we present tool support, using IR, ML, and RSSEs, that increases the level of automation in issue management. We define *automation* as proposed by Parasuraman *et al.*: "*a device or system that accomplishes (partially or fully) a function that was previously, or conceivably could be, carried out (partially or fully) by a human operator*" [367]. Parasuraman *et al.* further defines a model of ten levels of automation, presented in Table 1, starting from a fully *manual* process (Level 1) to an *automatic* process (Level 10). We refer to the activity of *automating* a process as increasing the level of automation. Moreover, the result of such an activity is an *automated* process. In this thesis, when we discuss automated IA and automated CIA, or automated tools in general, we refer to tools reaching Levels 3 or 4 in Parasuraman *et al*'s model. Thus, in contrast to some previous work in software engineering [23, 100, 293], we do not refer to automation below Level 10 as semi-automatic, even though we consistently require a human in the loop. We envision our automated tools to offer developers decision support, but the final action should still be executed by a human. The automation concept is further discussed in a related paper [VIII].

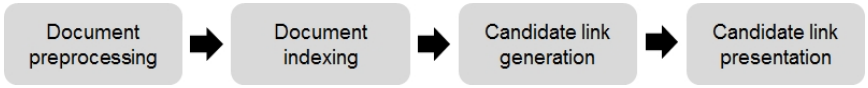


Figure 2: Key steps of an IR-based tracing tool, described by De Lucia *et al.* [145].

2 Related Work

This section presents selected publications related to the work in this thesis. We focus on three specific combinations of supporting techniques and software engineering work tasks: 1) IR for traceability management, 2) ML for issue assignment, and 3) RSSEs for navigation of software engineering information landscapes. As in Section 1, we keep the presentations short and point the reader to extended sections in the included papers.

2.1 Information Retrieval for Traceability Management

Several researchers have proposed implementing IR techniques in tracing tools, with the goal to help developers find or establish traces. The underlying idea is that if two development artifacts share a high degree of their textual content, they are more likely to be associated by a trace link. Figure 2 presents the key sequential steps involved in an IR-based tracing tool, based on a general description by De Lucia *et al.* [145]. First, the *documents are parsed and pre-processed*, i.e., the text is processed into tokens, and text operations such as stemming and stop-word removal might be applied. Second, an IR model is applied to *index the documents*, i.e., the documents are represented in a homogeneous document space. Third, the tracing tool compares a set of documents (representing trace artifacts) to *generate candidate trace links*. Fourth, the tracing tool *presents the candidate trace links* in the user interface.

Already in the early 1990s, researchers started proposing tool support for connecting software development artifacts containing NL text, e.g., in the LESD-project [81,83,99]. However, typically a publication by Fiutem and Antoniol from 1998 is considered the pioneering piece of work on IR-based tracing, in which the authors call their approach "*traceability recovery*", and they express the identification of trace links as an IR problem [183]. While their paper from 1998 applied only simple textual comparisons based on edit distances, their well-cited publication from 2002 instead used two traditional IR models: the Vector Space Model (VSM) and the binary independence model [18].

Since the pioneering work was published, several authors have continued working on IR-based tracing to support traceability management in software engineering projects. Natt och Dag *et al.* applied the VSM to support maintenance of dependencies among requirements in the dynamic environment of market-driven

requirements engineering [354]. Marcus and Maletic introduced latent semantic indexing to trace retrieval, and they applied it to recover trace links between source code and NL documentation [326]. Huffman Hayes *et al.* emphasized the user perspective in IR-based tracing, and introduced relevance feedback [233]. Cleland-Huang's research group have published several papers on probabilistic trace retrieval [307, 425, 498]. Finally, De Lucia *et al.* have introduced IR-based tracing in their document management system ADAMS [144]. More related work on IR-based tracing is presented in Paper II, reporting a comprehensive literature study on the topic, and in a complementary study [XV].

Most evaluations of IR-based tracing have been simplistic. First, a majority of the evaluations have been purely technology-oriented, conducted in what Ingwersen and Järvelin refer to as "*the cave of IR evaluation*" [237], i.e., not taking the user into account. Second, the size of the datasets studied in previous evaluations have been unrealistically small, typically containing fewer than 500 trace artifacts [Paper II]. Third, several evaluations have been conducted using trace artifacts originating from student projects instead of their industrial counterparts [IX]. Trace recovery evaluation was the topic of the licentiate thesis preceding this publication this, in which we argued that more evaluations in realistic user studies in industrial settings are needed to advance research on IR-based tracing [XI]. Consequently, our findings intensified CoEST's call for additional industrial case studies [203].

2.2 Machine Learning for Issue Assignment

In large development projects, the continuous inflow of issue reports might constitute a considerable challenge. Among the first tasks in issue management, the CCB must assign the issue to an appropriate developer or development team, see Figure 1. However, several studies report that manual issue assignment is tedious and error-prone, resulting in frequent reassignment of issue reports, so called "*bug tossing*", and delayed issue resolutions [44, 55, 248].

Several researchers have proposed automating issue assignment by introducing ML-based tool support. Figure 3 presents an overview of an automated issue assignment process. First, a *classifier* is trained on closed issue reports, using the developer that closed the issue as the label. Then, for future incoming issue reports, the classifier provides *decision support* to the CCB, i.e., the ML-based issue assignment proposes a suitable developer based on historical patterns. As developers continue to close issue reports, the classifier should be repeatedly *retrained*. Typically, previous work on ML-based issue assignment has represented issue reports by its NL text, i.e., the title and the description. A few exceptions include also nominal features available in issue trackers, e.g., submitter, priority, and platform [6, 308, 368].

Past research on ML-based issue assignment has evaluated several different classifiers. Čubranić *et al.* proposed using a Naïve Bayes (NB) classifier in a

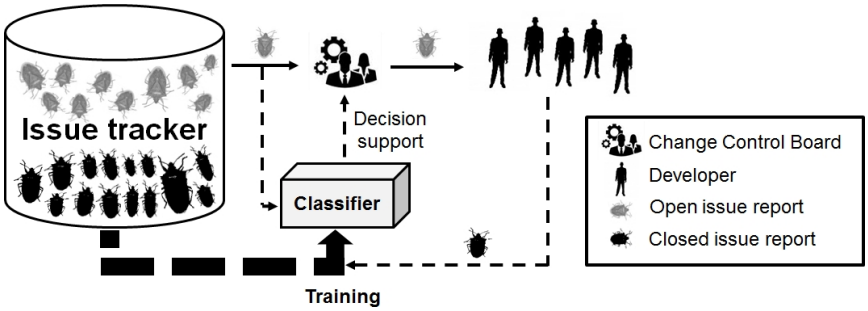


Figure 3: Overview of ML-based issue assignment. Solid arrows represent the current manual process, i.e., a CCB that assigns incoming issue reports. The dashed arrows depict the envisioned flow of automated assignment, i.e., a trained classifier provides the CCB with decision support in the assignment step.

pioneering paper [125]. NB has continued to be a popular technique for automated issue assignment, together with Support Vector Machines (SVM) as introduced by Anvik *et al.* [23]. However, while NB and SVM dominate in research, also other ML classifiers have been evaluated, e.g., random forest [6], Bayesian networks [248], and neural networks [217]. Paper III contains a comprehensive overview of related work on automated issue assignment.

Previous work on automated issue assignment has focused on Open Source Software (OSS) development projects, and especially issue reports from either the Eclipse development projects or the Mozilla Foundation. While the OSS context is meaningful to study, this thesis instead targets proprietary development projects. Regarding issue management, we highlight two important differences between proprietary and OSS projects. First, in OSS projects, anyone can typically submit issue reports to the issue tracker. In proprietary projects on the other hand, a majority of issue reports originate from internal development or testing activities. Thus, we hypothesize that the general quality of proprietary issue reports is higher. Second, proprietary development is typically organized in teams, but the organization of OSS projects is often less clear. Consequently, while previous evaluations of automated issue assignment have addressed assignment to *individual developers*, we instead evaluate assignment to *development teams*.

2.3 Recommendation Systems for Improved Navigation

The scale of the information landscape in large software engineering projects typically exceeds the capability of an individual developer [402]. Large software systems might evolve for decades [5, 170, 462], i.e., trace artifacts are continuously changing, introducing both versioning problems and obsolete information. Furthermore, with the increase of global software engineering, the producers of

information are distributed across different development sites. Consequently, an important characteristic of a software project is the *findability* it provides, i.e., "*the degree to which a system or environment supports navigation and retrieval*" [344].

Figure 4 presents four steps typically involved in RSSEs supporting navigation in software projects [XVI]. First, a *model of the information space* should be developed, describing all parts of the information landscape the RSSE covers. To express the relations among artifact types, work proposed in traceability research could be used, e.g., Ramesh and Jarke [384], Wnuk *et al.* [VII], or Rempel *et al.* [394]. Second, the *instantiated model need to be populated* to capture the trace artifacts and trace links in the information landscape, e.g., by mining software repositories [261]. Third, the RSSE *calculates recommendations* based on user actions. The calculations can be either triggered by the user explicitly, i.e., reactive initiation, or without any user invocation, i.e., proactive initiation [483]. Fourth, the RSSE must *present the recommendations* to the user. Murphy-Hill and Murphy list six approaches that previous RSSEs have used to deliver its output [346]: 1) annotations (textual markup), 2) icons, 3) affordance overlays (highlighting specific options), 4) pop-ups, 5) dashboards, and 6) e-mail notifications.

Several researchers have proposed RSSEs that alleviate information overload in software engineering. Čubranić *et al.* developed Hipikat, an RSSE that supports software evolution in OSS projects by helping newcomers come up-to-speed by establishing a "*project memory*" containing trace artifacts from various information repositories, e.g., the source code repository, the issue tracker, and the e-mail archive [127]. Hipikat then extracts explicit relations among software artifacts, and deducts additional trace links based on textual similarities. Finally, users interact with Hipikat through an Eclipse plug-in, and get recommendations presented in dedicated views. Maiga *et al.* proposed ReCRAC, an RSSE supporting both issue assignment and identification of similar issue reports for large software projects at Ericsson [323]. ReCRAC addresses the challenges of information overload by content-based filtering [XVI], but the tool has not yet been evaluated. Gethers *et al.* presented an approach to recommend change impact during software evolution in large software projects [192]. They combine IR techniques and analysis of execution information to recommend a candidate set of impacted source code, and they report promising results in a study on four OSS projects. Paper IV presents further examples of related work on RSSEs supporting navigation in software engineering.

Many RSSEs have been fully developed, but few have been evaluated in real software engineering projects. Robillard and Walker highlight the lack of user studies in the recent textbook on RSSEs [402]. They report that while recommendation algorithms can be analyzed in technology-oriented experiments, the only option to understand how developers react to recommendations is by conducting user studies. Also Tosun Misirli *et al.* stress the importance of studying RSSEs deployed in real projects, as the only way for researchers to "*understand the domain and propose technical solutions for real needs of practitioners*" [450, pp. 351].

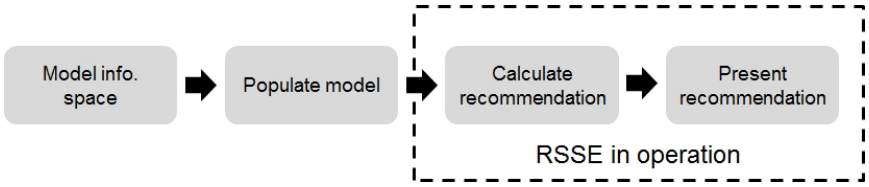


Figure 4: Principal steps of an RSSE for navigation support [XVI]. The dashed box indicates the two steps after deployment.

However, they acknowledge that deployment is hard, and that there are very few examples in the literature describing evaluations of real RSSE usage. Paper IV responds to the call from the RSSE community, by reporting from an *in situ* study of an RSSE deployed in a proprietary development project.

3 Research Overview

This doctoral thesis builds on the prequel licentiate thesis [XI]. The main contribution of the licentiate thesis was empirical evidence confirming the need for industrial case studies on IR-based tracing. In the licentiate thesis we presented a systematic mapping study (included as Paper II also in this doctoral thesis) showing that a majority of the previous evaluations were conducted *in silico* on datasets smaller than its industrial counterparts, and that the individual artifacts often originated from university settings. We also confirmed that most human-oriented evaluations were conducted *in vitro* with student subjects, i.e., in controlled classroom settings. Furthermore, to advance the quality of future evaluations, we proposed an evaluation taxonomy for IR-based tracing, an adaptation of Ingwersen and Järvelin's evaluation model for integrated information retrieval, introducing "*the cave of IR evaluation*" [237] as a concept in empirical software engineering.

While the licentiate thesis mainly involved exploratory work, this doctoral thesis contains evaluated solution proposals. As outlined in the licentiate thesis, we have continued working on a specific work task that requires developers to explicitly specify trace links among software artifacts: *Change Impact Analysis* (CIA) in safety-critical software development. We found initial evidence that developers are more comfortable navigating the source code than its related documentation, thus we focused work specifically on trace links between *non-code artifacts*. However, since the publication of the licentiate thesis, we have broadened the scope of our work on issue management. Thus, we do not only consider CIA, but also the initial *Issue Assignment* (IA), i.e., allocation of an issue report to the most appropriate development team.

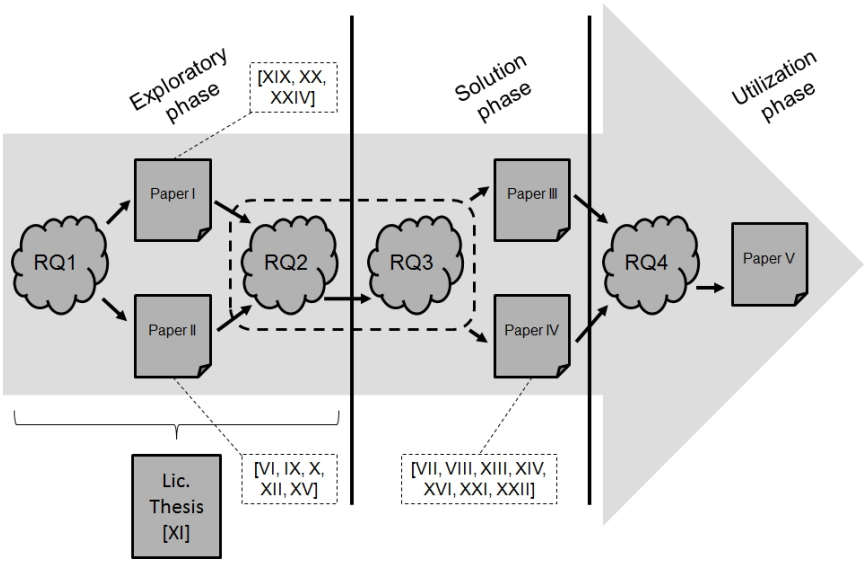


Figure 5: The three subsequent phases of research included in this thesis. The Exploratory phase concluded by an intermediate delivery of the licentiate thesis [XI] in September 2012. Since then, our work has turned more solution oriented, i.e., into the Solution phase and the Utilization phase. Dashed boxes with Roman numerals depict related publications.

The overall research goal of this thesis is to *leverage the information available in historical issue reports to support large-scale software evolution*. We further break down the research goal in the following research questions:

RQ1 How do state-of-practice approaches to maintain relations among development artifacts compare to proposals in the scientific literature?

RQ2 How can an increased level of automation support developers in issue management?

RQ3 How accurate do tools supporting issue management need to be for practitioners to start recognizing their value?

RQ4 How can transfer of research on automated tools for issue management to industry be supported?

Figure 5 shows an overview of the research included in this thesis, divided into three phases. The *Exploratory phase* was initiated by RQ1 and comprises several publications, whereof two papers are included in this thesis. Paper I reports

from an industrial case study on challenges and practices for alignment of requirements engineering and testing. As one of the identified challenges, we provide evidence that information access in state-of-practice software engineering projects is important but difficult, i.e., it is challenging for developers to stay on top of the continuously changing information landscape. Paper II instead targets state-of-the-art methods supporting maintenance of relations among development artifacts in software engineering projects, by reporting from a systematic mapping study. The exploratory phase was summarized in the licentiate thesis [XI], in which we outlined future work using solutions from Paper II to address challenges identified in Paper I (RQ2).

The *Solution phase* starts from RQ2 by posing the tightly connected RQ3, i.e., how good does tool support need to be to be useful in industry? In this thesis, the solution phase is represented by two papers that present tool support that we developed for automated issue management. Paper III considers IA as a classification problem, and evaluates an automated approach based on state-of-the-art ML. Paper IV proposes reusing historical traceability, i.e., the collaboratively created trace links, to support CIA triggered by corrective maintenance. We combine traceability analysis with state-of-the-art IR in an RSSE and evaluate the tool in an industrial case study.

The final *Utilization phase* is based on our observations that the correctness of our proposed tool support strongly depends on the specific development context, i.e., the outcome depends on the nature of the involved development artifacts (RQ4). Furthermore, we have developed tools that are highly configurable. Paper V attempts to match the two phenomena, i.e., the context dependency and the configurability, by presenting a framework for parameter tuning using state-of-the-art design of experiments. To support technology transfer to industry, Paper V presents a systematic approach to find a feasible parameter setting for automated tools. As a proof-of-concept, we apply the framework to optimize the correctness of the tool output from the RSSE in Paper IV, and discuss dangers of optimizing with regard to a single response variable. While Figure 5 illustrates the sequential work process shaping this thesis, successful technology transfer rather relies on iterative industry-academia collaboration [38, 201].

4 Research Methodology

Proper research strategies should be employed to pursue valid answers to the research questions. The research in this thesis is based on *empirical research*, a scientific approach to obtain knowledge through observing and measuring phenomena in the real world. Empirical studies result in evidence, pieces of information required to build and verify theories [167, 274, 413, 430]. This section gives an overview of the research methodologies applied in our empirical studies.

4.1 Purposes of Empirical Studies

Studies have different purposes, and there is not a single research strategy that fits them all. Runeson *et al.* list four purposes for research in software engineering [413], adapted from Robson [407]:

- **Exploratory** - discovering what is happening, pursuing new insights, and generating ideas and hypotheses for future research.
- **Descriptive** - characterizing the current status of a phenomenon or situation.
- **Explanatory** - seeking an explanation for a phenomenon, often in the form of a casual relationship.
- **Improving** - attempting to improve certain aspects of a phenomenon, and to evaluate the effect of improvement proposals.

The papers included in this thesis are either exploratory, covering work until the completion of the licentiate thesis [XI] (cf. Fig. 5), or of improving nature.

Exploratory research is typically conducted in early stages of a research project, to bring initial understanding to a phenomenon, preferably from rich qualitative data [167]. Exploratory studies are used to guide future work rather than to draw definite conclusions. Based on exploratory studies, informed decisions on for example the design of subsequent studies, data collection methods, and sampling strategies can be made. Papers I and II are both dominated by exploratory research. Paper I explored state-of-practice phenomena in industry, and Paper II summarizes state-of-the-art techniques presented in the scientific literature.

Improving research in software engineering tries to improve the current state-of-practice. As applied engineering researchers, we are inspired by the rather provoking quote by Theodore von Kármán (1881-1963): "*scientists study the world as it is, engineers create the world that has never been*". Thus, we strive to develop tools and processes to help practitioners develop higher quality software with less effort. An important part of improving research is the evaluation, i.e., the assessment of the effects and effectiveness of innovations, interventions, and practices [407]. The evaluative part involves a systematic collection of data, and a rigid analysis and interpretation. Paper III presents a solution for automated IA, and an evaluation based on more than 50,000 issue reports from two companies. Paper IV proposes automated support for CIA, and reports from an evaluation of the deployed tool. Also Paper V contains improving research, addressing the challenge of configuring tools for specific contexts.

4.2 Empirical Research Methods

When designing an empirical study, the researcher needs to find a suitable balance between the level of control and the degree of realism [413]. Studying phenomena in a real-world industrial context means less control of the involved variables,

and often there are too many confounding factors to conclude casual relationships. When isolating real-world phenomena on the other hand, e.g., by controlling subjects in lab environments, there is a risk that the software engineering aspect under study becomes reduced to something hardly representative of industrial reality [167].

Empirical research revolves around observations, and the collected data are either *quantitative* or *qualitative* [407]. Quantitative data constitute numbers obtained from measurements, and generally their purpose is to answer questions about the relationships between variables. For example, to quantify a relationship, comparing two or more groups, or for the purpose of explaining, predicting, or controlling a phenomenon. The researcher uses frequentist [129] or Bayesian [285] statistics to analyze the quantitative data: descriptive statistics to present the data, and inferential statistics to draw conclusions. Qualitative data involve words, descriptions, pictures etc. While quantitative data provide 'exactness', qualitative data instead offer 'richness' [413], enabling the researcher to understand a phenomenon beyond numbers. In software engineering research, qualitative data are often collected using interviews, enabling practitioners to express themselves in their own words. Analysis of qualitative data is based on the researcher's interpretation, and careful measures should be taken to mitigate biased conclusions. Researchers primarily rely on qualitative data when studying complex phenomena that cannot be simplified into discrete measurable variables. As quantitative and qualitative data are fundamentally different, studies typically reach the strongest conclusions by collecting and analyzing both kinds of data [422].

The design of empirical studies is often categorized as *fixed* or *flexible* [485]. Fixed designs are pre-specified and require enough pre-understanding of a phenomenon to know what to do, and how to measure it, already when the study is initiated. Studies relying on quantitative data are often of a fixed design. Flexible designs on the other hand, allow the study design to evolve while data is collected. The collection and analysis of data is intertwined, and both research questions and data sources may be adapted to the circumstances of the study. Paper I relies on a flexible design, and the study is based on qualitative data from interviews. Paper II used a fixed design, analyzing both quantitative and qualitative data collected from the scientific literature. Also Papers III and V employed fixed designs, analyzing quantitative tool output statistically. Paper IV on the other hand covers both a fixed and a flexible part, combining quantitative and qualitative data.

Empirical studies of both fixed and flexible designs can be conducted using different research methods. Easterbrook *et al.* consider five classes of research methods as the most important in software engineering [167]:

- **Experiments** - testing hypotheses by manipulating independent variables and measuring the effect on dependent variables.
- **Case studies** - investigating a contemporary phenomenon within its real-life context.

- **Surveys** - identification of population characteristics by generalizing from a sample.
- **Ethnographies** - understanding how a community of people make sense of their social interactions.
- **Action research** - attempting to solve a real world problem by intervention, while simultaneously studying the experience of solving the problem.

In this thesis, we neither use survey research nor ethnographies, but the other three methods are further described in the remainder of this section. However, we do not use the term "action research" to refer to our solution oriented work, but instead call it *design science*, as defined by Wieringa[3] [471]. Furthermore, one of the included papers in this thesis constitutes a secondary study following the guidelines for systematic literature reviews by Kitchenham and Charters [273].

Experiments (or controlled experiments) are used in software engineering to investigate the cause-effect relationships of introducing new methods, techniques or tools. Different treatments are applied to, or by, different subjects, while other variables are kept constant, and the effects on response variables are measured [477]. Wohlin *et al.* categorize experiments as either *technology-oriented* or *human-oriented*, depending on whether artifacts or human subjects are given various treatments. In this thesis, we use technology-oriented experiments to evaluate tool prototypes. As proposed by Walker and Holmes [466], we use *simulation* to conduct the technology-oriented experiments. In Papers III, IV, and V we use simulation to imitate the environment around the tools under evaluation, more specifically we reenact the historical inflow of issue reports to study the output of our automated tools. *Design of Experiments* (DoE) is a mature research field that has been applied to several application domains [342]. Paper V demonstrates how DoE can be used tune tools in software engineering with considerably less effort.

A *case study* in software engineering is conducted to understand a phenomenon within its real-life context. Such a study draws on multiple sources of evidence to investigate one or more instances of the phenomenon, and the research method is especially applicable when the boundary between the phenomenon and its context cannot be clearly defined. According to Runeson *et al.*, the case under study can be any contemporary software engineering phenomenon in its real-life setting, e.g., a group of people, a process, a specific role, or a technology [413]. Case studies are often conducted to explore a phenomenon, but they can also be confirmatory, i.e., designed to test existing theories [167]. Within a case study, researchers also distinguish between the case(s) and the unit(s) of analysis [485]. In a *holistic* case study, the case is studied as a whole. In an *embedded* case study on the other hand, multiple units of analysis are studied within a case, e.g., different development

[3] Wieringa discusses both *design science* and *Technical Action Research* (TAR) in his recent textbook [471], presenting TAR as an empirical method together with for example surveys and experiments.

sites, development teams, or roles in the organization. Moreover, a case study can be characterized as a *single-case* study, or a *multiple-case* study if two or more cases are studied within different contexts [413]. Paper I is categorized as an exploratory embedded single-case study [413, pp. 185], i.e., the case (alignment of requirements engineering and testing) is studied in multiple units of analysis (six different companies). Also Paper IV is an embedded single-case study, where the CIA work task constitutes the case and two different development teams are the units of analysis. Moreover, as Paper IV also evaluates the usefulness of a tool, the case study has both exploratory and confirmatory aspects.

A *Systematic Literature Review* (SLR) is a secondary study aimed at aggregating a base of empirical evidence. SLRs are an increasingly popular method in software engineering research [131]. Inspired by evidence-based medicine, SLRs rely on a rigid search and analysis strategy to ensure identification of a comprehensive collection of evidence related to a specific question [273]. A variant of an SLR is a *Systematic Mapping Study*, a literature study designed to identify research gaps and direct future research [273, 373]. Paper II is a systematic mapping study targeting IR-based tracing, representing one possible approach to support maintenance of large information spaces in software engineering.

Design science, a problem solving paradigm, is defined by Wieringa as "*the design and investigation of artifacts in context*" where an artifact is "*something created by people for some practical purpose*" [471]. We apply design science methodology to conduct improving empirical research, i.e., we strive to create innovative artifacts to improve the software engineering state-of-practice. According to Wieringa, the two major components in a design science research project are the *design activity* and the the *investigative activity*. The design activity builds on understanding of the context to develop innovations that support an explicit need. The investigative activity seeks to understand the interaction between the artifact and the context, e.g., to investigate the needs of stakeholders, or to evaluate a deployed artifact. Design science is an iterative process, and Hevner refers to the interplay between the design activity and the investigative activity as the build-evaluate loop [219]. The two activities alternate during repeated iterations in the loop until the results are satisfactory. Papers III and IV involve design science, as we develop tool support for automated issue management in close collaboration with industry partners. The design activities are based on identified needs in the the target contexts, and we conduct several iterations of the build-evaluate loop as the tools evolved.

4.3 Settings of Empirical Studies

Software engineering experiments are sometimes classified as either *in vivo* or *in vitro* [42], depending on the experimental setting and the trade-off between control and realism. *In vitro* represents experiments conducted in an environment where a high level of control can be attained, whereas studies in an *in vivo* setting

instead are executed in a real-world setting, also known as the field. But this traditional two-tier classification scheme offers only a coarse-grained separation of experimental settings in empirical software engineering. For example, Travassos and Barros argue that the increased use of computer models in software engineering strongly influences the experimentation process, and that this variation is not properly highlighted in the traditional *in vivo/in vitro* classification [451].

While working on a secondary study, Travassos and Barros introduced two additional levels to the classification: *in virtuo* and *in silico*. The extension, inspired by classification schemes used in biology [223], resulted in a four-tier classification. Experiments conducted *in virtuo* involve a virtual environment, composed by computer models that represent real-world elements or phenomena. In such studies, researchers let human subjects interact with the virtual environment. In the *in silico* setting on the other hand, experiments are completely executed using computer models. Again the environment is virtual, but also the interaction with the environment is simulated using computer models (instead of human subjects). The benefits of *in virtuo* and *in silico* experiments is that they require considerably less effort, and that they are easier to replicate. Travassos and Barros even propose a pool of "*virtual software engineers*" that can be used for pilot runs before any real-world experiments take place [451].

The description of Travassos and Barros' four-tier classification scheme is only intended for human-oriented experiments. The discussion provided by Travassos and Barros on *in silico* experiments does not match the type of software engineering experiments referred to by Wohlin *et al.* as technology-oriented [477], i.e., without human subjects. On the contrary, Travassos and Barros claim that "*in silico studies are still very rare in software engineering*" [451]. However, numerous technology-oriented software engineering experiments have been published, and also this thesis includes such empirical enquiries in Papers III, IV, and V. Thus, to better suit the classification of papers included in this thesis, we define technology-oriented experiments as conducted in an *in silico* setting. *In virtuo* studies on the other hand, do not apply to this thesis, and we do not discuss them further.

To enable classification of all papers in this thesis, we also consider case studies. Case studies, i.e., an "*investigation of a contemporary phenomenon within its real-life context*" [413], can by definition not be conducted in a controlled or simulated setting. However, we argue that the *in vivo* setting should be complemented by an *in situ* setting. A researcher can study a contemporary phenomenon in its real-life context by letting practitioners share experiences in interviews, and the setting can be referred to as *in vivo*. To refer to the setting as *in situ* on the other hand, the phenomenon should be studied in its real-life when it happens, e.g., using ethnographies [405] or using instrumenting systems [295]. Note that the *in situ* setting has been mentioned in previous publications on research methodology in software engineering, e.g., Perry *et al.* [372] and Kitchenham *et al.* [275], but we explicitly distinguish it from *in vivo*.

To conclude, in this thesis we use the following four-tier classification scheme

of study settings, ordered by an increasing level of realism:

- **In silico** studies are conducted entirely on a computer. The object under study is a software artifact and the researcher analyzes its output. Sometimes the study is designed as a technology-oriented experiment [477], in less rigid studies *in silico* evaluations rather act as proofs-of-concept, demonstrating the feasibility of an approach. *In silico* evaluations have successfully advanced state-of-the-art techniques in IR, ML, and RSSEs, e.g., using benchmarking [415] in the format of contests [119] such as in the TREC experiments for IR [434].
- **In vitro** studies are executed with human subjects in a controlled environment. In software engineering, most *in vitro* studies are human-oriented experiments in a university lab [451]. Often it is too costly to involve practitioners in the experiments, instead student subjects are frequently studied in classroom settings [225].
- **In vivo** studies explore software engineers in their own environments. Typically we refer to practitioners working in private or public software development organizations, but it can also be used for studies of developers working from home in open source software projects. Case studies based on interviews [413, pp. 50], possibly triangulated with analysis of archival data, constitute examples of studies in an *in vivo* setting.
- **In situ** studies seek to understand software engineers in their own environments, in real-time as they conduct their work tasks. To accomplish the real-time data collection, the researcher must use what Lethbridge *et al.* call first or second degree data collection methods [295], i.e., the researcher is in direct contact with the software engineers, or the researcher directly collects raw data from a remote location. Example approaches for *in situ* studies include long-term observational studies [278] and instrumenting systems [295].

Finally, for simplicity, we define that literature studies are conducted in an orthogonal **secondary** setting [273].

4.4 Classification of Included Papers

Table 2 summarizes the research strategy applied in this thesis. For each of the included papers, we list: 1) the addressed research question, 2) the purpose of the study, 3) the research method applied, and 4) the setting of the study. The rest of this section further presents and motivates our classification.

Papers I and II address how relations among software artifacts in large software engineering projects are maintained. Paper I is an exploratory *in vivo* case study in which we interviewed 30 practitioners in six companies. The goal of the

Table 2: Classification of papers included in the thesis, and a mapping to the research questions.

Paper	RQ	Purpose	Main research methods	Setting
I	RQ1	Exploratory	Single case multi-unit case study	*In vivo*
II	RQ1	Exploratory	Systematic mapping study	Secondary
III	RQ2, RQ3	Improving	Design science, technology-oriented experiments	*In silico*
IV	RQ2, RQ3	Improving	Design science, single case two-unit case study	*In situ*
V	RQ4	Improving	Technology-oriented experiment	*In silico*

study was to get a deep understanding of the current state of alignment between Requirements Engineering and Testing (RET) in industry practice. We define the case as *RET alignment in software engineering practice*, and each of the six studied companies is considered a unit of analysis. While other perspectives could be taken, e.g., enabling a comparative view by considering the study as consisting of six cases as discussed by Runeson *et al.* [413, pp. 185], we prefer to pool the 30 interviews for a rich picture of state-of-practice challenges and practices related to RET alignment. Paper II instead explores the state-of-the-art in the scientific literature regarding one specific approach to support maintenance of relations among development artifacts, namely IR-based tracing. We identified 79 primary studies and classified them based on the type of development artifacts linked, the IR techniques applied, and the strength of the corresponding empirical evaluations.

RQ2 concerns how an increased level of automation can support issue management in large software engineering projects. RQ3 is closely related, expressing how accurate automated tools need to be to actually support software engineers. We target both RQ2 and RQ3 by two studies with improving purposes, i.e., a large part of the involved work follows a design science methodology [471]. In Paper III, we propose increasing the level of automation for IA in proprietary contexts by training ensemble classifiers. Also, we implement the approach in a tool and present an evaluation in the form of technology-oriented experiments *in silico*. In Paper IV, we discuss providing software engineers with automated decision support for CIA using the RSSE *ImpRec*. While we present an *in silico* study on ImpRec, we primarily evaluate the RSSE in an *in situ* case study. Two development teams constitute the units of analysis, and we study how the software engineers interact with ImpRec as part of their work in the field. In both Papers III and IV, we provide comparisons with current manual approaches and discuss what is needed by automated tools in issue management, as well as what risks are introduced by an increased level of automation.

Finally, RQ4 deals with technology transfer, an important final step for applied

researchers. Our work shows that the accuracy of automated issue management is highly dependent on the specific development context, i.e., the nature of the involved development artifacts appears to influence the result more than the selected technique. Thus, tuning the automated tool support for the specific deployment context is fundamental to reach the potential of a tool. Paper V, also with an improving purpose, describes how to use design of experiments [342] to systematically, *in silico*, find feasible parameter settings for software engineering tools.

5 Results

This section introduces the included papers, with a particular focus on findings related to the four RQs. For each paper, we explain the rationale behind the study, and the empirical data behind the conclusions. We summarize the main contributions of the papers in bullet lists, and conclude by describing how the paper is related to our parallel work, not included in this thesis.

Paper I: An Industrial Case Study on Alignment of Requirements Engineering and Testing

Both requirements engineering and testing aim to support cost-efficient development of software systems, meeting the customers' demands on functionality and quality. However, while both requirements engineering and testing constitute mature research communities in software engineering, few studies have focused on the alignment of the "two ends of software development". Paper I provides deep insights in the current state-of-practice of Requirements Engineering and Testing (RET) alignment through interviews with practitioners. Figure 6 captures our preunderstanding of the RET phenomenon in a conceptual model, a starting point for the discussions with all interviewees in the study.

We conducted a large case study following a rigorous process involving 11 researchers. In total, we interviewed 30 practitioners representing different roles related to RET (e.g., requirements engineers, testers, and project managers) from six companies in Sweden and Norway. The interviews were guided by three RQs exploring: 1) current challenges in RET alignment, 2) current practices supporting RET alignment, and 3) which challenges are addressed by which practices. The main contributions of Paper I are:

- Empirical evidence of *16 challenges in RET alignment*, including:
  - Tracing between requirements and test cases.
  - Managing a large document space.
  - Maintaining alignment when requirements change.

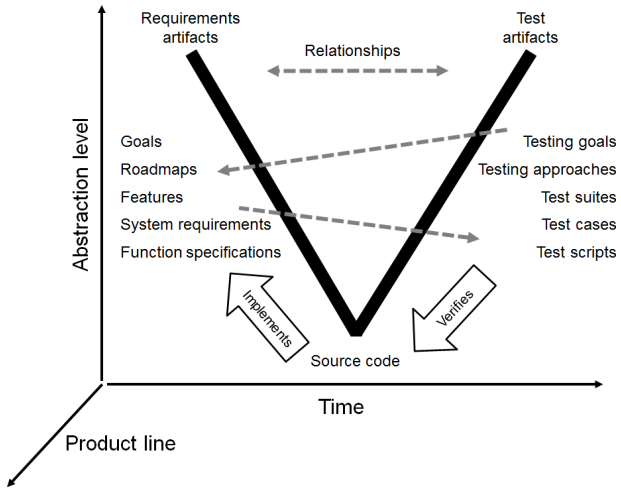


Figure 6: Conceptual model of RET alignment. While the illustration resembles a single iteration of the traditional V-model, we consider also iterative development, i.e., "a series of consecutive Vs", in our conceptual model.

- Identification of *27 practices that support RET alignment* in industry, including:
  - Maintaining trace links between development artifacts.
  - Using test cases as requirements.
  - Initiating testing activities early in the development process.
- A mapping of the practitioners' views of *how the identified practices address the RET challenges*, providing actionable guidance for long-term process improvement in industry.
- Four high-level observations on RET alignment:
  1. *The human side is central*, i.e., the communication and collaboration between people is vital.
  2. *The quality of the requirements is critical* for the subsequent testing activities.
  3. *The size of the development organization is a key variation factor* in what challenges arise and what practices are applied.
  4. Practices involving *requirements documentation and tracing are applied for development projects mandated by external process requirements* (e.g., safety standards), but only to a limited extent in organi-

zations with nothing but internal motivation on formal documentation and traceability.

Paper I has opened an avenue for future research on RET alignment. The paper laid the foundation for the RET workshop series[4]. Due to the size of the study, the practical undertaking provided valuable experiences from running a large-scale case study involving 11 researchers and more than 210,000 words in the interview transcripts. Experiences and lessons learned have been reported in a separate chapter in a textbook on case study research by Runeson *et al.* [413, pp. 183-199]. Based on the findings from Paper I, we have so far published two additional papers: 1) an experience report on to what extent the identified challenges apply to a case of large-scale software development in the public sector in Sweden [XIX], and 2) a deeper analysis of one of the RET practices identified, i.e., using test cases as requirements [XXIV]. Finally, two subsets of authors have developed theoretical frameworks to support RET research and practice. Bjarnason *et al.* used the interview data to inductively generate a theoretical model of 'gaps' that obstruct RET alignment [63], and Unterkalmsteiner *et al.* developed a taxonomy to describe and assess RET alignment in an organization [456].

Paper II: A Systematic Mapping Study of IR-based Tracing Tools

One state-of-practice approach to structure the challenging document space in large software development projects, confirmed by Paper I, is to maintain traceability from requirements to later artifacts. However, a number of previous studies have shown that maintaining trace links among artifacts is both tedious and error-prone. To support traceability management, several researchers have proposed IR-based tracing to propose candidate trace links between artifacts with a high degree of textual similarity. Paper II surveys existing research on IR-based tracing, and reports to what extent the approach has been applied to maintain traceability both between requirements and test artifacts, and development artifacts in general.

We conducted a comprehensive literature review of a decade's worth of IR-based tracing research. Our goal was to identify both technical and empirical aspects of previous work, i.e., both which IR models that had been implemented, and the strength of evidence from previous evaluations with regard to feasibility in an industrial setting. Using an established methodology for systematic literature reviews, we aggregated empirical data from 132 studies reported in 79 publications. The main contribution from the literature study is evidence that:

- A majority of research on IR-based tracing implemented algebraic IR models, most often the classic VSM with cosine similarities.

[4]International Workshop on Requirements Engineering and Testing (RET'14) [XX] (RET'15)

- Most studies address trace links either: 1) between different requirements abstraction levels, or 2) between requirements and source code (cf. Fig. 7). A few studies discuss tracing between requirements and test cases, but there is potential for future work.
- A handful of controlled experiments with student subjects suggest that IR-based tracing tools help engineers establish correct trace links faster, e.g., in the context of CIA.
- Most previous evaluations provide low strength of evidence, and case studies in industry are much needed. For example:
  - Most studies do not analyze the usefulness of IR-based tracing further than tool output, i.e., *in silico* evaluations conducted "*in the cave*" [237] dominate.
  - A majority of the evaluations have been conducted in simplified document spaces, most often involving fewer than 500 artifacts, and often the artifacts originate from small student projects.

We mapped the primary studies identified in Paper II onto the levels of the evaluation taxonomy we proposed in prior work [X], and the outcome was used as the starting point for two subsequent studies. First, we further investigated the *in silico* results reported in the primary studies [XV]. We show that no IR model regularly outperforms the traditional VSM, and we instead present results indicating a strong data dependency, i.e., the dataset under study influences the experimental results more than the choice of IR model. Second, we conducted a qualitative survey among the authors of the primary studies, regarding the validity of using artifacts collected from student projects for evaluation of IR-based tracing [IX]. Our results show that a majority of authors consider student artifacts to be only partly representative to industrial artifacts, but still the student artifacts are rarely validated for realism prior to using them in tracing experiments. Finally, based on our observations of different link types depicted in Figure 7, we hypothesize that issue reports could be considered important information hubs in the information landscape, constituting junctures between RE and testing artifacts (cf. Fig. 7). This hypothesis paved the way for the focus on issue reports permeating Papers III, IV, and V.

Paper III: Experiments on Automated Issue Assignment

In large software development projects the inflow of issue reports in a project can be daunting, threatening the engineers' ability to stay on top of the situation. To be able to use issue reports to support navigation between RE and testing artifacts, the overall issue management process needs to be efficient and effective. One of the challenges involved in early issue triaging is IA, i.e., allocating appropriate developers to resolve the issue. Several researchers have proposed automated IA using

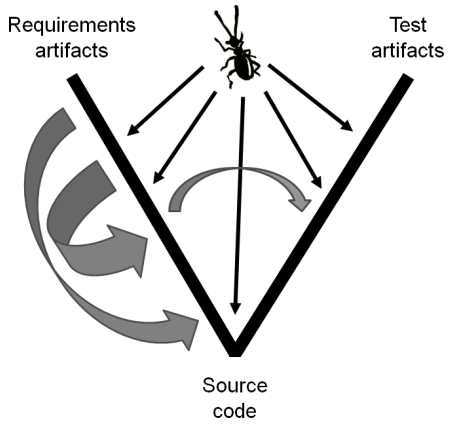


Figure 7: Types of trace links in research on IR-based tracing. Most studies target traces between requirements on different abstraction levels, or between requirements and source code. Fewer studies address tracing between requirements and test artifacts. We propose using issue reports, illustrated by a bug, as informations hubs connecting different parts of the conceptual model.

ML, but the previously developed tools have almost exclusively been evaluated in the context of OSS projects. Paper III explores to what extent results from the OSS domain can be transferred to the proprietary contexts of two of our industry partners.

We developed a prototype for automated IA using stacked generalization, i.e., a state-of-the-art ML approach to combine individual classifiers into an ensemble. To evaluate the feasibility in proprietary contexts, we collected historical issue reports from five projects originating from two industry partners. We conducted controlled experiments varying ensemble selections and the amount of training data, and the main contributions of the paper are:

- A prototype for automated IA, based on OSS components, that is easy to deploy in state-of-practice issue repositories.
- The largest evaluation of ML-based issue *team* assignment in proprietary contexts, in contrast to assignment to *individual developers* in OSS projects, based on more than 50,000 issue reports from industry.
- Initial evidence that the prototype, without context-specific tuning, has the potential to assign issue reports, in a fraction of a second, with an accuracy in line with current manual work.
- An empirically grounded rule-of-thumb: *at least 2,000 issue reports should be included in the training set*. However, *continuous monitoring of the pre-*

diction accuracy is fundamental, as the prediction accuracy deteriorates as older training data is used, i.e., the benefit of adding more issue reports in the training might be nullified by the disadvantage of training on less recent data.

Besides IA, we have explored ML-based support for several other decisions involved in early issue triaging. Inspired by Paper IV, we have defined and supervised three MSc thesis projects. First, Johansson explored increasing the number of features representing an issue report [250], and applying learning-to-rank ranking [311], to *improve duplicate detection among issue reports*. However, our efforts did not outperform the off-the-shelf IR system Apache Lucene using default settings [XXI]. Second, Olofsson and Gullin developed a prototype ML-tool at Sony Mobile Communications that with an accuracy of 70% *predicted whether an issue report would lead to a corrective code change or not* [361]. Third, Mauritzon used latent dirichlet allocation to *identify patterns in very large-scale issue reports from the customer support organization* at Sony Mobile Communications [333]. All three MSc thesis projects indicate that an increased level of automation in the issue inflow, based on ML, has the potential to provide large organizations with actionable decision support.

Paper IV: An Industrial Case Study on ImpRec: An RSSE for Change Impact Analysis

Once an issue report has been assigned to the appropriate development team, and the need for a corrective code change has been confirmed, a developer should assess how the change will affect the rest of the software system. Especially in safety-critical development contexts CIA is a fundamental activity, mandated by various safety standards as a formal activity that must be completed prior to changing any source code. Not only do developers need to specify how other pieces of source code are impacted, but also how requirements might be influenced, whether any of the system's quality attributes will be affected, and what test cases should be re-executed to verify the changes etc. Several studies report that conducting a CIA can be a cumbersome and error-prone activity in complex software systems, and that practitioners mostly conduct CIA without tool support.

We propose supporting CIA by establishing a knowledge base of previous impact, i.e., a collaboratively created network of trace links, connecting historical issue reports and various development artifacts in the information landscape. We implement the approach in the RSSE ImpRec, a prototype tool for automated CIA, tailored for an organization developing safety-critical automation systems. We evaluate ImpRec in a two-step industrial case study, consisting of a static validation followed by a dynamic validation *et al.* [200]. The main contributions of the case study, and the involved design science, are:

- A prototype RSSE for automated CIA, combining state-of-the-art IR with a collaboratively created knowledge base of trace links.
- A longitudinal *in situ* evaluation of ImpRec deployed in two separate development teams, a type of human-oriented research that has been much neglected within RSSE research [402, pp. 9].
- A quantitative assessment of the ImpRec *correctness*, i.e., the static validation, complemented by a qualitative evaluation of the ImpRec *utility*, i.e., the dynamic validation, two quality dimensions proposed by the RSSE community [32].
- A discussion, based on the QUPER model [49], on how accurate RSSE output for CIA needs to be for developers to consider it useful.

The development of ImpRec started before the case study reported in Paper IV was initiated. While ImpRec evolved considerably during the case study, the backbone development of ImpRec was presented in a previous chapter in an edited book on RSSEs [XVI]. The book chapter in turn builds on three publications. First, we showed the presence of extensive networks of issue reports, consisting of related issues, in the public Android issue repository and a proprietary counterpart [XIV]. Based on this finding, we argued that network analysis could be applied to improve tools supporting search and navigation in software projects. Second, we suggested link mining in historical CIA reports to establish a semantic network of system traceability [XIII]. Also, we provided initial suggestions on how the trace links in the semantic network could be (re-)used in tool support for CIA. Third, in a study on duplicate detection, we showed that Apache Lucene, a state-of-the-art OSS IR solution, provides a promising approach to identify similar issue reports [XXI]. The implementation of ImpRec combines elements from the three publications, namely: 1) network analysis to support search and navigation, 2) link mining to enable traceability reuse, and 3) scalable searching for similar issue reports using Apache Lucene. Parallel to the case study in Paper IV, we conducted a survey with practitioners to better understand the challenges involved in state-of-practice CIA [XXII]. The survey covers several industry sectors, e.g., automotive, aerospace, and railroad, thus we show that many challenges identified in the automation context generalize to other domains, such as: *insufficient tool support* and *a vast number of artifacts to trace*.

Paper V: Tuning Software Engineering Tools Using Design of Experiments

Software engineering tools are often highly configurable through parameters. While the parametrization allows context-specific tuning, it also requires the users to find a feasible parameter setting. Several studies show that default settings might not

let tools reach their potential. However, as software engineering tools often implement advanced techniques, finding a feasible parameter setting often constitutes a considerable challenge.

Paper V describes how carefully designed experiments can be used to tune software engineering tools. We present TuneR, a three-phase tuning framework using R, combining space-filling designs and response surface methodology. As an illustration of how to apply TuneR, and as a proof-of-concept of the framework, we use it to tune ImpRec to the safety-critical context presented in Paper IV. The main contributions of the study are:

- A tuning framework for software engineering tools with extensive guidelines and a hands-on example.
- A discussion on how traditional Design of Experiments (DoE) and Design of Computer Experiments (DoCE) differ from experiments intended to tune software engineering tools.
- A proof-of-concept tuning of ImpRec, resulting in a 21% increase in the response variable as compared to the initial *ad hoc* tuning during ImpRec development.
- Using an exhaustive approach, we show that a 24% improvement would have been possible. However, the exhaustive approach requires more than 50 hours of computing time, and do not provide any insights in how the parameter interact.

During the work on this thesis, we experienced how much the performance of a tool depends on the context, and the need for easy-to-follow tuning guidelines for software engineering tools became evident. The strongest piece of evidence comes from the Paper II follow-up study [XV], in which we show that the dataset under study appears to influence the results of IR-based tracing more than the selected IR model. However, also in our other tool-oriented studies, we noticed that one single parameter setting typically does not provide results that generalize to all contexts. The observed importance of parameter settings is in line with work by other tool researchers, e.g., Arcuri and Fraser [25] and Thomas *et al.* [446], and we argue that tuning research could support tool adaptation in industry.

6 Synthesis

This section provides answers to the four research questions, by synthesizing the results from the included papers.

RQ1: How do state-of-practice approaches to maintain relations between development artifacts compare to proposals in the scientific literature?

Paper I reports from a broad case study on RET alignment in industry practice. While the scope of the study goes beyond maintenance of artifact relations, some findings are closely related. For example, practitioners in three of the six studied companies explicitly report "managing a large document space" as a RET challenge. The interviewees report that the sheer number of development artifacts makes it difficult to get an overview, as sometimes tens or hundreds of thousands of requirements, test cases, and other documents constitute the information landscape in large software engineering projects. The information overload is further complicated by constant modifications of development artifacts in evolving systems, changes that might lead to redundant information, in line with Wnuk *et al.*'s observation of obsolete requirements [475]. Some interviewees in Paper I also stress that the information management in software engineering projects is obstructed by inadequate tool support. For example, updating information can be cumbersome, and tools turn into "information silos" due to poor interoperability. A state-of-practice approach to structure large information spaces is to maintain explicit trace links between artifacts, but several interviewees report that connecting artifacts in different databases is tedious manual work, especially in projects with large legacies. The challenge of maintaining trace links is well-known in the literature [202], and our findings from Paper I provide further evidence. Still, in development contexts regulated by strict process standards (e.g., IEC 61511 [238] and ISO 26262 [241]), both backward and forward traceability must be maintained through the entire product lifecycle.

Several researchers have proposed supporting trace link management by increasing the level of automation through tool support. Based on the assumption that development artifacts that are textually similar are likely to be related, a family of tools based on IR approaches have evolved. Paper II shows that parts of the traceability research community have been highly active in research on IR-based tracing, and tools and approaches have been proposed and improved for more than a decade. However, we show that the strength of evidence supporting the usefulness of IR-based tracing is low, as no tools have been evaluated in large *in situ* studies. We also argue that a purely academic race for marginally better tool output brings little practical value, especially since the current generation of tracing tools appear to have plateaued [114]. Our sceptical position toward minor tool improvements resonates with findings from experiments by Cuddeback *et al.* [128] and Dekhtyar *et al.* [151]; even though the correctness of a tracing tool's output improves, it does not necessarily lead to better overall decisions by the human working with the tool. While we have also touched upon this phenomenon in a pilot experiment [VI], only further user studies can provide holistic understanding of the interaction between humans and tracing tools.

Based on Papers I and II we conclude that *state-of-practice management of artifact relations in information spaces include management of explicit trace links*, and that *several researchers have proposed introducing IR-based tools to support the tracing*. However, there is still *a considerable gap between industry practice and the research proposed by the traceability community*. Apart from a case study on a small development organization in China [304], there are no success stories reporting utility of IR-based tracing in industry. Instead, the strongest evidence in support of IR-based tracing tools comes from experiments with student subjects in classroom settings (e.g., De Lucia *et al*. [148] and Huffman Hayes *et al*. [232]). To close the gap between research and industry, we stress that *there is a strong need to perform industrial case studies to understand how, and if, developers can benefit from IR-based tracing*. Thus, we reinforce CoEST's call for additional empirical work [202].

RQ2: How can an increased level of automation support developers in issue management?

The constant inflow of issue reports in large software development projects can be daunting, putting developers in a state of information overload. Most previous research on tool support for issue management has focused on the inflow of issue reports in OSS development projects, where typically anyone can submit reports in public issue trackers. In this thesis, we show that also in proprietary projects, with internal issue management processes, the inflow of issue reports might constitute a considerable challenge. Even though the obvious "junk reports" that sometimes plague OSS projects (e.g., general nonsense, spam, and even worms [52]) are rare in proprietary contexts, also the volume of valid issue reports can be obstructive.

In Papers III and IV, we explore the effect of increasing the level of automation in the issue management process by providing additional decision support. We target two specific work tasks involved in issue management, both highlighted as challenging by our industry partners. First, we address IA, the activity of allocating appropriate developers to resolve an issue. In some projects, frequent re-allocations, referred to as "bug tossing", causes both delayed issue resolutions and general frustration among developers [55, 248]. Second, we target CIA, the task of assessing the impact of software changes on the rest of the system. CIA is a fundamental activity in development and evolution of safety-critical software systems, but several studies report it to be costly and error-prone [68, 208, 271].

The overall approach we advocate in this thesis, the backbone of both Papers III and IV, is to *guide issue management based on historical patterns in the issue tracker*. We posit that there is potential to provide actionable decision support based on a project's history of issue reports, and that it is wasteful to not pay attention to the ever-growing amount of information available in issue trackers. While the last decade saw numerous publications describing automated tool support along those lines, the number of studies reporting success stories from

industry remains low. On the other hand, our position concurs with the ongoing expansion of data-driven decision support and business intelligence in various domains [256, 380], and we argue that software development projects should be no exception.

Finding patterns in issue trackers appears to be particularly promising for supporting navigation of large project landscapes. Previous studies argue that the issue tracker of a project acts as a collaborative hub [24, 100], effectively connecting both developers and development artifacts. Both Papers III and IV report successful evaluations of tools that leverage on historical issue reports to provide decision support in issue management, reaching Levels 4 and 3 as defined by Parasuraman [367] (cf. Table 1). In Paper III, we use ML to *train classifiers on historical issue reports* from five proprietary development projects, and *demonstrate increased automation in team assignment of incoming issue reports*. In Paper IV, we describe how an *RSSE can reuse a proprietary project's history*, captured in a collaboratively created knowledge base, *to recommend potential impact* when implementing corrective changes due to incoming issue reports.

RQ3: How accurate do tools supporting issue management need to be for practitioners to start recognizing their value?

To confirm whether tool support actually supports developers in industry, studies must consider what happens when human users enter the picture. Avazpour *et al.* discuss evaluation of RSSEs from different perspectives by defining 16 quality dimensions [32], e.g., correctness, novelty, usability, and scalability. In this thesis, we focus on two of the quality dimensions: *correctness* and *utility*. Correctness compares how close an RSSE's output is to the output that is assumed to be correct, and utility measures how much value a user actually gains from the RSSE's output. While correctness is a quality dimension that can be assessed using *in silico* studies, understanding utility requires human-oriented studies. However, as highlighted in Paper II, there are few published user studies on IR-based tracing tools. Moreover, the lack of user studies generalizes to both other aspects of traceability research [202] and evaluations of RSSEs in general [402].

In Papers III and IV, we map the correctness of the proposed tools to the utility of the tools. In both papers, we discuss utility based on the QUPER model [49]. The QUPER model considers quality to be a continuous, but non-linear, characteristic with three distinguishable breakpoints: 1) utility, 2) differentiation, and 3) saturation. In this thesis, we are particularly interested in whether the proposed tools have passed the *utility breakpoint*, i.e., do developers working with our tools recognize their value?

Paper III argues that the correctness, discussed in terms of prediction accuracy, of an automated tool does not have to be as high as for a fully manual process to bring value in IA, since the automated process provides candidate assignments

in a fraction of a second. Furthermore, we envision that deployed automated IA supports issue management by providing decision support, but does not fully replace the human decision making. However, preliminary measurements of bug tossing in one of the studied organizations indicate that *the correctness of the automated approach is in line with the current manual process*, thus we claim that our approach has passed the utility breakpoint. Future work should be directed at deploying the tool in an operational setting, to fully understand the potential of automated IA.

Paper IV reports from a more direct mapping of correctness and utility. First we measured the correctness of our RSSE using *in silico* simulations, re-enacting the historical inflow of issue reports, by conducting automated CIA for a representative subset of the inflow. The simulations yielded correctness similar to what has been reported in related work on automated CIA, i.e., about 40% of the true impact is presented among the top-20 recommendations. Reassured by the simulation results, we deployed the RSSE in two development teams in a longitudinal *in situ* study. As a final step in the study, we conducted interviews with developers to collect their experiences. Again we based discussions on the QUPER model, and we conclude that *the additional search and navigation provided by ImpRec, at the current level of correctness, corresponds to a tool that has passed the utility breakpoint.*

During the work on this thesis, we have also identified other examples of research on automated issue management that have transferred to state-of-practice tools. Bugzilla[5] has support for duplicate detection of issue reports at submission time. While the user types the title of a new issue report, Bugzilla concurrently presents possible duplicates. HP Quality Center also provides automated duplicate detection of issue reports, and the feature is explicitly used in the marketing of the tool [220]. Furthermore, Cavalcanti *et al.* report that nine other issue trackers implement IR techniques for duplicate detection, including JIRA and Trac [100]. However, the authors conclude that the advanced techniques presented in research papers have not yet been adopted in state-of-practice issue trackers, an observation that has also been reported in a literature study by Zhang *et al.* [493]. Both Cavalcanti *et al.* and Zhang *et al.* hypothesize that the results reported in state-of-the-art research might not be sufficient to convince industry practitioners. In May 2012 on the other hand, a US patent with the title "System and method for maintaining requirements traceability" was granted [48], describing an approach to synchronize artifacts during software evolution. While the actual tracing process is only described in general terms, a research paper implementing IR-based tracing is one of few academic publications cited [314]. To conclude, we argue that the examples of initial technology transfer mentioned above indicate that at least parts of the research on automated issue management is mature enough for industrial use, as for example rudimentary duplicate detection has disseminated to state-of-practice issue trackers.

[5] http://www.bugzilla.org/

RQ4: How can transfer of research on automated tools for issue management to industry be supported?

Developing research prototypes is not the end goal of software engineering research; we should also consider how to transfer new technology to industry practice [201]. A frequent concern in studies on prototype tools, often discussed as a threat to validity, is the scalability of various approaches. For the tools proposed in Papers III and IV, we have developed solutions with the industrial scale in mind. The techniques involved in this thesis typically perform better as more data is available, i.e., we leverage on the scale of the problem. In this thesis, we address another obstacle to successful technology transfer of tools to industry: to properly adapt them to a specific operational setting.

The correctness of output from software engineering tools is typically context dependent. Often advanced algorithms are implemented that rarely have a single setting that yields the best results in all contexts [479]. We have highlighted the evident data dependency for IR-based tracing tools in a follow-up study of Paper II, showing that the dataset used for evaluating a tool might influence the correctness more than the selected IR model [XV]. Also within an operational context the performance of a tool might vary over time, due to changes in either the software under development or the development process applied. We discuss this phenomenon in Paper III, and stress the importance of continuously monitoring the quality of the tool output, as sudden decreases might indicate that an ML-based approach should disregard parts of the available training data.

To help tackling the challenge of variations in operational contexts, many tools proposed by software engineering researchers are highly configurable. However, researchers cannot expect practitioners to master the underlying mechanisms of their tools, thus practitioners need support for both initial tool deployment and subsequent maintenance. In Paper III we apply stacked generalization [478], an approach to combine multiple classifiers in an ensemble. While the idea of the ensemble is to provide a more robust ML-based tool for IA, there are still several variation points that need to be considered before deployment in industry, e.g., which classifiers to include in the ensemble, and how to preprocess the textual data. In Paper IV we present ImpRec, an RSSE with a configurable ranking function for candidate impact. The parameters of the ranking function set relative weights of the influence of the textual similarity and the network measures, but finding a feasible setting is not straight-forward.

Several studies show that relying on the default settings of a tool might lead to suboptimal performance [25, 58, 313], and we argue that *there is a need for tuning guidelines for software engineering tool support*. In Paper V, written in a tutorial style, *we introduce TuneR: an experimental framework for tuning software engineering tools*. Based on established research on design of experiments [342], successful in numerous engineering applications [236], we discuss how to adapt the experimental approach to better suit the tuning of software engineering tools.

TuneR combines space-filling designs and response surface methodology in a three phase framework, and as a proof-of-concept we apply TuneR to improve the correctness of ImpRec by 20.9%.

Finally, Paper IV includes a preliminary discussion on another aspect of technology transfer: the *usability* of the tool support. Usability, also defined as a quality dimension by Avazpour *et al.* [32], implies that a tool should be both effective and efficient, as well as providing some degree of satisfaction for their target users. Interviews with developers regarding experiences with ImpRec confirm the importance of providing tool support that is easy to use, and that lowering thresholds that inhibit tool use is critical. Some highlighted aspects of usability include *true integration into the existing issue tracker*, *fast tool queries*, and *intuitive options to filter output*. The importance of information filtering is in line with our previous finding from focus groups on tool support for regression test scoping, stressing that interaction with tool output is central [XVII].

7 Threats to Validity

The results of any research effort should be questioned, even though proper research methodologies were applied. The validity of the research is the foundation on which the trustworthiness of the results is established. We discuss the primary validity threats related to our RQs, and the actions taken to reduce the threats, using the classification proposed by Runeson *et al.* [413]. Based on our research goals, we emphasize external validity rather than internal validity. At the same time, construct validity and reliability were maximized given the resources available during the studies.

Construct validity is concerned with the relation between theories behind the research and the observations. Consequently, it covers the choice and collection of measures for the studied concepts. For example, the questions during an interview must not be misunderstood or misinterpreted by the interviewee. Two strategies to increase construct validity are: 1) using multiple sources of evidence, and 2) establishing chains of evidence [485].

RQ1 is partly answered through a large interview study [Paper I], a study that involved 11 researchers and 30 interviewees. While the selection of interviewees covered six companies, there is still a risk that we captured a limited or unbalanced view of the construct, i.e., RET alignment. Two other construct validity threats apply to our interview study, as well as to interview studies in general. First, academic researchers and industry practitioners might use different terminology, leading to misunderstandings. Second, the interview situation might influence the interviewee to either hide facts or to respond after assumed expectations, a phenomenon called *reactive bias*. We mitigated these threats during the design of the study, by iteratively developing an interview guide that was both reviewed and piloted, as well as by producing a conceptual model of RET alignment that

was discussed early in the interviews to align the terminology (cf. Fig. 6). To reduce the reactive bias, we did not offer any rewards for participation in the study, and all interviewees were guaranteed anonymity. The other part of RQ1, the state-of-the-art review, is addressed by a systematic mapping study. We argue that the identification of primary studies is subject to threats to construct validity, as the search string we use must match the terminology used by traceability researchers. To mitigate this threat, we validated the search string on a 'gold standard' set of publications, established by a combination of database searches and snowball sampling [476].

RQ3 is primarily discussed based on the QUPER model [49]. In Paper III we use the model in our argumentation regarding the utility breakpoint, and in Paper IV we use it directly in interviews with practitioners to stimulate discussions on QUPER's three quality breakpoints. We consider two threats to using QUPER in these ways as particularly important. First, QUPER was developed to support roadmapping of quality requirements in market-driven requirements engineering, but we use the model to discuss the value of automated tools. Second, the interviewees might not have properly understood the QUPER model, or our approach to map the quality of tool output to its breakpoints. To mitigate these threats, the interview guide used in Paper IV was reviewed, with a focus on understandability, by several researchers including the principal QUPER developer. Furthermore, a previous evaluation of QUPER describes the model as "*not very easy [to understand], but definitely not difficult*" [49], and the interviewees in Paper IV did also not signal any considerable problems.

Internal validity is related to issues that may affect the causal relationship between treatment and outcome. In experiments, the internal validity refers to whether the effect is caused by the independent variables or other confounding factors. To minimize the effect of confounding factors, experimental designs typically employ randomization, replication, and blocking [342]. In a case study reporting a successful tool evaluation, an example internal validity threat is work processes that change during the data collection, i.e., the observed improvements might have been caused by process improvements rather than additional tool support. For descriptive and exploratory studies however, internal validity is typically not a threat as casual claims rarely are made [485].

The four RQs introduced in this chapter are all of exploratory nature, as indicated by the 'how' formulations. Consequently, we do not consider confounding factors a primary validity threat. However, causality is indirectly involved, as Papers III, IV, and V to various extent report results from experiments. An example threat to the internal validity in Paper III is that we use default settings for all classifiers studied, a choice that possibly favors some of them, thus the ensemble selections we conclude as the best in the different study contexts might have been influenced. Moreover, Paper II is a secondary study aggregating experimental results from previous work, thus our conclusions are affected by the threats to the primary studies. However, in the light of the four 'how' RQs stated in this chapter,

the absence of casual claims limits the threats to internal validity of our corresponding conclusions. Further discussions on internal validity are available in the individual papers.

External validity concerns the ability to generalize the findings outside the actual scope of the study, i.e., results obtained in a specific context may not be valid in other contexts. While quantitative studies can apply statistics and sampling strategies to strengthen generalizability, qualitative studies must rely on analytical generalization [413, pp. 71], i.e., researchers can extend results to other cases that have similar characteristics. To support analytical generalization from a qualitative study, it is essential to present a comprehensive context description [374]. Other strategies to address threats to external validity include studying multiple cases and replicating experiments [485].

Related to RQ2, we propose two tool solutions to increase the level of automation: ML-based IA and an RSSE for CIA. The first tool is evaluated using more than 50,000 issue reports from five projects in industry. While the issue reports originate from only two proprietary contexts developing embedded software systems, both the content of the issue reports and the team structures are similar to what we have observed in other companies. Furthermore, our approach is in line with previous work evaluated in OSS projects and we have not tuned the tools to the specific projects. To conclude, we consider the external validity of our approach to automated IA as high. The second tool on the other hand, implemented in the tool ImpRec, is tailored for one specific development organization. ImpRec's knowledge base is mined from the issue tracker in that organization, and obviously the tool, in its current form, can only be used in this context. However, as safety standards from different domains share many aspects of traceability and CIA, we argue that the overall approach, i.e., to enable reuse of previously established trace links [XIII], has potential to support also other safety-critical development contexts.

Also regarding the other three RQs, it is worthwhile to question the generalizabilty of our findings. RQ1 explores state-of-practice RET alignment in six different companies, but we focus on one single state-of-the-art approach to manage artifact relations, i.e., maintaining trace links among software artifacts. However, there are several other approaches to support engineers in large software engineering information landscapes, e.g., model-based engineering [158], non-IR tools supporting traceability management [113], and general process improvement [286]. RQ3 addresses how good automated solutions in issue management need to be for practitioners to recognize their value. We exclusively discuss the utility breakpoint of our two tool solutions, and it is possible that our findings do not extend to other related approaches to increase the level of automation in issue management, e.g., severity prediction [290], effort estimation [XXVI], and fault localization [389]. Finally, within the scope of RQ4, we propose a tuning framework to support technology transfer. The tuning framework has only been applied for ImpRec, thus it should be evaluated also for other automated solutions. More-

over, there are other obstacles to technology transfer than the tuning issue, e.g., large-scale deployment of tool support [164], and the challenge of conformance to existing development processes [30]. To further support technology transfer to industry, we should address such other obstacles in future work.

Reliability relates to whether the same outcome could be expected by another set of researchers, both in terms of what data was collected and how the data was analyzed. For qualitative studies, the analysis relies on the interpretation of the researchers, thus exact replications are highly improbable. During a qualitative study, researchers should be aware of, and critically confront, favored lines of interpretation [12]. An example threat to reliability is a poorly designed case study protocol, obscuring the practical steps involved in data collection and analysis.

The state-of-practice analysis of RQ1 is purely based on qualitative research, i.e., interview studies with practitioners, thus the reliability of our findings should be questioned. Our approach to increase reliability is based on the number of researchers involved in the study. To reduce the influence of individual researchers, all findings in the study, as well as all steps leading to that conclusion, were reviewed by at least one other researcher. Furthermore, the research process was systematically documented in the case study protocol; a living document during the entire research endeavor. Nonetheless, another set of researchers might have reported other challenges, e.g., less emphasis could potentially by placed on the tracing of artifacts by researchers with different preunderstandings.

Also RQ2 and RQ3 are subject to threats to reliability, as we primarily answer them by synthesizing findings from the included papers. Synthesizing evidence from other studies, especially qualitative results, also involves a high degree of interpretation [124]. We propose automated solutions for issue management related to two challenges highlighted by our industry partners; other researchers might, in collaboration with their industry partners, decide to focus on other topics. An important concern relates to the general dangers of automation; each increase in the level of automation introduces new risks [367]. In particular, in Paper IV we could have investigated this further, and it is possible that other researchers would claim that introducing additional tool support in safety-critical development contexts is generally unwise. Also, regarding RQ3, the discussions related to the utility breakpoint in the QUPER model are based on our interpretations. To fully understand how much practitioners benefit from automated tools, and whether the current level of correctness offered by the tools in this thesis is indeed helpful, future studies of tools deployed in real operational settings must be conducted.

8 Conclusion and Future Work

The overall goal of this thesis is to leverage the information available in historical issue reports to support large-scale software evolution. To work toward this goal, we have conducted research in three sequential phases: 1) the exploratory phase,

2) the solution phase, and 3) the utilization phase. All publications included in this thesis build on empirical research, primarily case study research and experiments.

The exploratory phase is represented by two papers investigating information management in software engineering, covering current state of industry practice and the state of research contributions, respectively. In Paper I, based on a case study of six companies, we show that it is *difficult for developers to stay on top of the highly dynamic information landscapes of large software engineering projects*. We also show that one of the applied practices to maintain structure in a software engineering project, especially in safety-critical development contexts, is to *manage explicit trace links among development artifacts*. However, trace link management is known to be labor-intensive, thus in response several researchers have developed tracing tools implementing Information Retrieval (IR) techniques. Paper II reports from a systematic mapping study of IR-based tracing and shows that while 79 papers were published before 2011, the number of evaluations in industry is suspiciously low. Consequently, we conclude that there is a *considerable gap between the state-of-practice and approaches suggested by the scientific community*, and that *additional case studies in industry are needed to close the gap* (RQ1). Guided by these conclusions, we focused the remainder of the thesis project on case studies and experiments in real world proprietary software engineering contexts.

The solution phase builds on the exploratory phase by using techniques from Paper II to address difficulties identified in Paper I, and more specifically, challenges related to information access in large software engineering projects. Focusing on the constant inflow of issue reports, *we present automated tools that leverage on the volume of historical issue reports to deliver actionable decision support* (RQ2), and we evaluate our proposed tools using more than 60,000 issue reports from proprietary projects in industry. Paper III proposes ensemble-based Machine Learning (ML) for Issue Assignment (IA). Paper IV introduces a Recommendation System for Software Engineering (RSSE), *ImpRec*, which uses a collaboratively created knowledge base to support non-code Change Impact Analysis (CIA) in a safety-critical development context. In the solution phase, we also explore whether the proposed tools provide utility to the users (RQ3). Our results in Paper III indicate utility, as our *ensemble learner performs in line with the current manual process*, but the IA is completed without time-consuming manual analysis. In Paper IV, we evaluate ImpRec both *in silico*, i.e., in technology-oriented experiments, and *in situ* by deploying the RSSE in two development teams. Based on discussions with study participants around the QUPER model, we conclude that *the additional search and navigation provided by ImpRec*, correctly identifying about 50% of the non-code change impact within the top-20 recommendations, *provides a useful starting point during CIA*.

The final phase in this thesis, the utilization phase, addresses how to support transfer of research results on automated issue management to industry (RQ4). In agreement with previous work, we have experienced that *the accuracy of au-*

tomated tools strongly depends on the operational context, i.e., the nature of the development artifacts in the specific software engineering project. Furthermore, automated tools implementing techniques based on IR, ML, or RSSEs are typically highly configurable through parameters. However, researchers cannot expect practitioners to understand all the details required to obtain a feasible parameter setting. Consequently, to support the tuning of automated tools to a specific operational context, Paper V *presents guidelines in the form of TuneR: an experimental framework for tuning software engineering tools*. TuneR packages research knowledge in a form that allows advanced practitioners to tune automated tools for their specific contexts. As a proof-of-concept, we apply TuneR to improve the correctness of ImpRec by 20%.

In conclusion, this thesis presents how an increased level of automation in issue management can support tasks central to software evolution. We show that approaches leveraging historical issue reports are mature enough to support practitioners in industry. Moreover, by conducting studies beyond "the cave", i.e., outside the comfort zone of academic evaluations on *de facto* benchmarks in the lab, we show that tool performance clearly depends on the data involved in the operational setting. The empirical evidence we present regarding data dependency should have implications for future research on automated tools; the practical value of minor tool improvements on specific datasets, despite statistically significant results, could be debated due to uncertain generalizability. In this thesis, we address the challenge of data and context dependency by presenting TuneR, a tuning framework that helps releasing the potential of software engineering tool support.

Work on this thesis has opened several potential directions for future work. First and foremost, we stress that there is a need for further research in connection with the utilization phase. Analogous to recommendations from other researchers, i.e., both the RSSE community [402] and the traceability community [202], more *in situ* research with users is needed to understand how automated issue management can best be applied in industry, in line with the study we present in Paper IV. There are several important questions that have not yet been explored in literature, even though they are critical to successful technology transfer. First, how can tool support that increases the level of automation in issue management be deployed in an organization without jeopardizing developers' ability to think beyond the recommendations? Two possible research directions to tackle this challenge are: 1) tailoring work processes based on the increased level of automation, and 2) developing methods to deliver the recommendations in ways that stimulate critical thinking. Second, once the automated system has been deployed, how should it be maintained? As presented in Paper III, the prediction accuracy might deteriorate over time, thus it is critical to continuously monitor the output quality. But how could the tools best be retrained, or reconfigured, when a significant change has been detected? For an automated system to be successful over time in industry, it must be developed with maintainability in consideration, as industry practitioners cannot be assumed to be experts in advanced techniques such as IR and ML.

Avazpour *et al.* refer to this quality aspect of an RSSE as *stability* [32].

Another avenue for future work based on the contributions of this thesis involves technical aspects of automated issue assignment. Previous research reveals that academic researchers tend to focus on such aspects, thus we expect advances in a number of areas. First, research on automated issue management typically studies a static dataset of development artifacts, i.e., "batch learning" from a one-shot dump of part of the informations landscape of a software engineering project. In a real setting, the inflow of additional development artifacts, and the constant evolution of existing artifacts, introduces questions on how RSSEs should be practically updated. Thus, future work should investigate *online learning* for RSSEs [226], to develop automated tools that do not require complete retraining as new training data becomes available. Second, future RSSEs might better incorporate direct feedback from the user, in line with work by Čubranić *et al.* on Hipikat [127]. By enabling quick and easy feedback through the user interface, an RSSE would be able to adjust recommendations faster than through the traditional re-training loop presented in Figure 3. *Relevance feedback* is well-known in IR [325], and those ideas have already been employed in IR-based tracing tools [146, 154, 282]. Third, the correctness of an RSSE cannot go beyond the quality of the features used to represent the software artifacts in the information landscape. Thus, we anticipate that many of the advances in research on automated tools, as well as successful applications in the field, will come after significant *feature engineering*, i.e., identifying, integrating, cleaning, and preprocessing of artifact properties. In accordance with lessons learned in ML communicated by Domingos [161], our experiences acknowledge feature engineering as being the key to successful application of the family of tools discussed in this thesis.

The next generation of issue trackers should incorporate decision support to learn from the constant inflow of issue reports. The ever-increasing volume of issue reports is not merely a burden; tool developers should embrace the bugs and harness their rich information, collected as issue reports move through different states during the resolution process. Recommendation systems are commonplace for numerous applications on the web, and everyone with an online presence has received years of training on assessing the value of automated suggestions. It is time for software engineering to leverage the bug inflow, both to increase developers' confidence by confirming existing views, i.e., that an issue is likely to imply "business as usual", and to detect anomalies among the incoming issue reports. While several important research questions remain to be explored, we argue that tool developers should already start looking at automated decision support for issue management. Anything else would be a waste of good bugs.

PART I: THE EXPLORATORY PHASE

CHAPTER I

CHALLENGES AND PRACTICES IN ALIGNING REQUIREMENTS WITH VERIFICATION AND VALIDATION: A CASE STUDY OF SIX COMPANIES

Abstract

Weak alignment of requirements engineering (RE) with verification and validation (VV) may lead to problems in delivering the required products in time with the right quality. For example, weak communication of requirements changes to testers may result in lack of verification of new requirements and incorrect verification of old invalid requirements, leading to software quality problems, wasted effort and delays. However, despite the serious implications of weak alignment research and practice both tend to focus on one or the other of RE or VV rather than on the alignment of the two. We have performed a multi-unit case study to gain insight into issues around aligning RE and VV by interviewing 30 practitioners from 6 software developing companies, involving 10 researchers in a flexible research process for case studies. The results describe current industry challenges and practices in aligning RE with VV, ranging from quality of the individual RE and VV activities, through tracing and tools, to change control and sharing a common understanding at strategy, goal and design level. The study identified that human aspects are central, i.e. cooperation and communication, and that requirements engineering practices are a critical basis for alignment. Further, the size of an organisation and its motivation for applying alignment practices, e.g. external

enforcement of traceability, are variation factors that play a key role in achieving alignment. Our results provide a strategic roadmap for practitioners improvement work to address alignment challenges. Furthermore, the study provides a foundation for continued research to improve the alignment of RE with VV.

Elizabeth Bjarnason, Per Runeson, Markus Borg, Michael Unterkalmsteiner, Emelie Engström, Björn Regnell, Giedre Sabaliauskaite, Annabella Loconsole, Tony Gorschek, and Robert Feldt, *Empirical Software Engineering*, 19(6), pp. 1809-1855, 2014.

1 Introduction

Requirements engineering (RE) and verification and validation (VV) both aim to support development of products that will meet customers' expectations regarding functionality and quality. However, to achieve this RE and VV need to be aligned and their 'activities or systems organised so that they match or fit well together' (MacMillan Dictionary's definition of 'align'). When aligned within a project or an organisation, RE and VV work together like two bookends that support a row of books by buttressing them from either end. RE and VV, when aligned, can effectively support the development activities between the initial definition of requirements and acceptance testing of the final product [133].

Weak coordination of requirements with development and testing tasks can lead to inefficient development, delays and problems with the functionality and the quality of the produced software, especially for large-scale development [283]. For example, if requirements changes are agreed without involving testers and without updating the requirements specification, the changed functionality is either not verified or incorrectly verified. This weak alignment of RE and work that is divided and distributed among engineers within a company or project poses a risk of producing a product that does not satisfy business and/or client expectations [198]. In particular, weak alignment between RE and VV may lead to a number of problems that affect the later project phases such as non-verifiable requirements, lower product quality, additional cost and effort required for removing defects [414]. Furthermore, Jones *et al.* [252] identified three other alignment related problems found to affect independent testing teams, namely uncertain test coverage, not knowing whether changed software behaviour is intended, and lack of established communication channels to deal with issues and questions.

There is a large body of knowledge for the separate areas of RE and VV, some of which touches on the connection to the other field. However, few studies have focused specifically on the alignment between the two areas [39] though there are some exceptions. Kukkanen *et al.* reported on lessons learnt in concurrently improving the requirements and the testing processes based on a case study [286]. Another related study was performed by Uusitalo *et al.* who identified a set of practices used in industry for linking requirements and testing [458]. Furthermore,

RE alignment in the context of outsourced development has been pointed out as a focus area for future RE research by Cheng and Attlee [108].

When considering alignment, traceability has often been a focal point [39, 365, 468]. However, REVV alignment also covers the coordination between roles and activities of RE and VV. Traceability mainly focuses on the structuring and organisation of different related artefacts. Connecting (or tracing) requirements with the test cases that verify them support engineers in ensuring requirements coverage, performing impact analysis for requirements changes etc. In addition to tracing, alignment also covers the interaction between roles throughout different project phases; from agreeing on high-level business and testing strategies to defining and deploying detailed requirements and test cases.

Our case study investigates the challenges of RE and VV (REVV) alignment, and identifies methods and practices used, or suggested for use, by industry to address these issues. The results reported in this paper are based on semi-structured interviews of 90 min each with 30 practitioners from six different software companies, comprising a wide range of people with experience from different roles relating to RE and VV. This paper extends on preliminary results of identifying the challenges faced by one of the companies included in our study [414]. In this paper, we report on the practices and challenges of all the included companies based on a full analysis of all the interview data. In addition, the results are herein categorised to support practitioners in defining a strategy for identifying suitable practices for addressing challenges experienced in their own organisations.

The rest of this paper is organised as follows: Section 2 presents related work. The design of the case study is described in Section 3, while the results can be found in Section 4. In Section 5 the results are discussed and, finally the paper is concluded in Section 6.

2 Related Work

The software engineering fields RE and VV have mainly been explored with a focus on one or the other of the two fields [39], though there are some studies investigating the alignment between the two. Through a systematic mapping study into alignment of requirements specification and testing, Barmi *et al.* found that most studies in the area were on model-based testing including a range of variants of formal methods for describing requirements with models or languages from which test case are then generated. Barmi *et al.* also identified traceability and empirical studies into alignment challenges and practices as main areas of research. Only 3 empirical studies into REVV alignment were found. Of these, two originate from the same research group and the third one is the initial results of the study reported in this paper. Barmi *et al.* draw the conclusions that though the areas of model-based engineering and traceability are well understood, practical solutions including evaluations of the research are needed. In the following sections pre-

vious work in the field is described and related to this study at a high level. Our findings in relation to previous work are discussed in more depth in Section 5.

The impact of RE on the software development process as a whole (including testing) has been studied by Damian *et al.* [134] who found that improved RE and involving more roles in the RE activities had positive effects on testing. In particular, the improved change control process was found to 'bring together not only the functional organisation through horizontal alignment (designers, developers, testers and documenters), but also vertical alignment of organisational responsibility (engineers, teams leads, technical managers and executive management)' [134]. Furthermore, in another study Damian and Chisan [133] found that rich interactions between RE and testing can lead to pay-offs in improved test coverage and risk management, and in reduced requirements creep, overscoping and waste, resulting in increased productivity and product quality. Gorschek and Davis [198] have proposed a taxonomy for assessing the impact of RE on, not just project, but also on product, company and society level; to judge RE not just by the quality of the system requirements specification, but also by its wider impact.

Jointly improving the RE and testing processes was investigated by Kukkanen *et al.* [286] through a case study on development performed partly in the safety-critical domain with the dual aim of improving customer satisfaction and product quality. They report that integrating requirements and testing processes, including clearly defining RE and testing roles for the integrated process, improves alignment by connecting processes and people from requirements and testing, as well as, applying good practices that support this connection. Furthermore, they report that the most important aspect in achieving alignment is to ensure that 'the right information is communicated to the right persons' [286, pp. 484]. Successful collaboration between requirements and test can be ensured by assigning and connecting roles from both requirements and test as responsible for ensuring that reviews are conducted. Among the practices implemented to support requirements and test alignment were the use of metrics, traceability with tool support, change management process and reviews of requirements, test cases and traces between them [286]. The risk of overlapping roles and activities between requirements and test, and gaps in the processes was found to be reduced by concurrently improving both processes [286]. These findings correlate very well with the practices identified through our study.

Alignment practices that improve the link between requirements and test are reported by Uusitalo *et al.* [458] based on six interviews, mainly with test roles, from the same number of companies. Their results include a number of practices that increase the communication and interaction between requirements and testing roles, namely early tester participation, traceability policies, consider feature requests from testers, and linking test and requirements people. In addition, four of the companies applied traceability between requirements and test cases, while admitting that traces were rarely maintained and were thus incomplete [458]. Linking people or artefacts were seen as equally important by the interviewees who

were unwilling to select one over the other. Most of the practices reported by Uusitalo *et al.* were also identified in our study with the exception of the specific practice of linking testers to requirements owners and the practice of including internal testing requirements in the project scope.

The concept of traceability has been discussed, and researched since the very beginning of software engineering, i.e. since the 1960s [386]. Traceability between requirements and other development artefacts can support impact analysis [134, 204, 286, 385, 458, 468], lower testing and maintenance costs [286, 468], and increased test coverage [458,468] and thereby quality in the final products [385, 468]. Tracing is also important to software verification due to being an (acknowledged) important aspect in high quality development [385, 468]. The challenges connected to traceability have been empirically investigated and reported over the years. The found challenges include volatility of the traced artefacts, informal processes with lack of clear responsibilities for tracing, communication gaps, insufficient time and resources for maintaining traces in combination with the practice being seen as non-cost efficient, and a lack of training [111]. Several methods for supporting automatic or semi-automatic recovery of traces have been proposed as a way to address the cost of establishing and maintaining traces [144,232,316]. An alternative approach is proposed by Post *et al.* [379] where the number of traces between requirements and test are reduced by linking test cases to user scenarios abstracted from the formal requirements, thus tracing at a higher abstraction level. When evaluating this approach, errors were found both in the formal requirements and in the developed product [379]. However, though the evaluation was performed in an industrial setting the set of 50 requirements was very small. In conclusion, traceability in full-scale industrial projects remains an elusive and costly practice to realise [204,245,383,468]. It is interesting to note that Gotel and Finkelstein [204] conclude that a 'particular concern' in improving requirements traceability is the need to facilitate informal communication with those responsible for specifying and detailing requirements. Another evaluation of the traceability challenge reported by Ramesh identifies three factors as influencing the implementation of requirements traceability, namely environmental (tools), organisational (external organisational incentive on individual or internal), and development context (process and practices) [383].

Model-based testing is a large research field within which a wide range of formal models and languages for representing requirements have been suggested [158]. Defining or modelling the requirements in a formal model or language enables the automatic generation of other development artefacts such as test cases, based on the (modelled) requirements. Similarly to the field of traceability, model-based testing also has issues with practical applicability in industrial development [341, 356, 487]. Two exceptions to this is provided by Hasling *et al.* [214] and by Nebut *et al.* [356] who both report on experiences from applying model-based testing by generating system test cases from UML descriptions of the requirements. The main benefits of model-based testing are in increased test cov-

erage [214, 356], enforcing a clear and unambiguous definition of the requirements [214] and increased testing productivity [207]. However, the formal representation of requirements often results in difficulties both in requiring special competence to produce [356], but also for non-specialist (e.g. business people) in understanding the requirements [317]. Transformation of textual requirements into formal models could alleviate some of these issues. However, additional research is required before a practical solution is available for supporting such transformations [487]. The generation of test cases directly from the requirements implicitly links the two without any need for manually creating (or maintaining) traces. However, depending on the level of the model and the generated test cases the value of the traces might vary. For example, for use cases and system test cases the tracing was reported as being more *natural* than when using state machines [214]. Errors in the models are an additional issue to consider when applying model-based testing [214]. Scenario-based models where test cases are defined to cover requirements defined as use cases, user stories or user scenarios have been proposed as an alternative to the formal models, e.g. by Regnell and Runeson [392], Regnell *et al.* [393] and Melnik *et al.* [336]. The scenarios define the requirements at a high level while the details are defined as test cases; acceptance test cases are used to document the detailed requirements. This is an approach often applied in agile development [94]. Melnik *et al.* [336] found that using executable acceptance test cases as detailed requirements is straight-forward to implement and breeds a testing mentality. Similar positive experiences with defining requirements as scenarios and acceptance test cases are reported from industry by Martin and Melnik [330].

3 Case Study Design

The main goal of this case study was to gain a deeper understanding of the issues in REVV alignment and to identify common practices used in industry to address the challenges within the area. To this end, a flexible exploratory case study design [407, 413] was chosen with semi-structured interviews as the data collection method. In order to manage the size of the study, we followed a case study process suggested by Runeson *et al.* [413, chapter 14] which allowed for a structured approach in managing the large amounts of qualitative data in a consistent manner among the many researchers involved. The process consists of the following five interrelated phases (see Fig. 1 for an overview, including in- and outputs of the different phases):

1. *Definition* of goals and research questions
2. *Design and planning* including preparations for interviews
3. *Evidence collection* (performing the interviews)

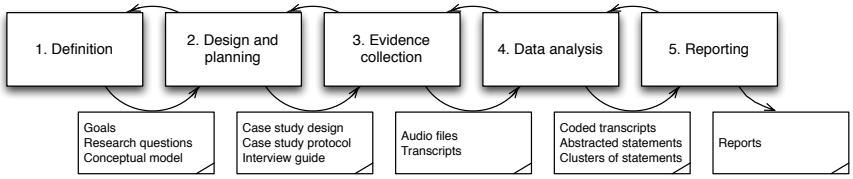


Figure 1: Overview of the research process including in- and output for each phase.

4. *Data analysis* (transcription, coding, abstraction and grouping, interpretation)

5. *Reporting*

Phases 1-4 are presented in more detail in Sections 3.1 to 3.4, while threats to validity are discussed in Section 3.5. A more in-depth description with lessons learned from applying the process in this study is presented by Runeson *et al.* [413, chapter 14]. A description of the six case companies involved in the study can be found in Section 3.2.

The ten authors played different roles in the five phases. The senior researchers, Regnell, Gorschek, Runeson and Feldt lead the *goal definition* of the study. They also coached the *design and planning*, which was practically managed by Loconsole, Sabaliauskaite and Engström. *Evidence collection* was distributed over all ten researchers. Loconsole and Sabaliauskaite did the transcription and coding together with Bjarnason, Borg, Engström and Unterkalmsteiner, as well as the preliminary *data analysis* for the evidence from the first company [414]. Bjarnason, Borg, Engström and Unterkalmsteiner did the major legwork in the intermediate *data analysis*, coached by Regnell, Gorschek and Runeson. Bjarnason and Runeson made the final *data analysis, interpretation* and *reporting*, which was then reviewed by the rest of the authors.

3.1 Definition of Research Goal and Questions

This initial phase (see Fig. 1) provided the direction and scope for the rest of the case study. A set of goals and research questions were defined based on previous experience, results and knowledge of the participating researchers, and a literature study into the area. The study was performed as part of an industrial excellence research centre, where REVV alignment was one theme. Brainstorming sessions were also held with representatives from companies interested in participating in the study. In these meetings the researchers and the company representatives agreed on a main long-term research goal for the area: to improve development efficiency within existing levels of software quality through REVV alignment, where

this case study takes a first step into exploring the current state of the art in industry. Furthermore, a number of aspects to be considered were agreed upon, namely agile processes, open source development, software product line engineering, non-functional requirements, and, volume and volatility of requirements. As the study progressed the goals and focal aspects were refined and research questions formulated and documented by two researchers. Four other researchers reviewed their output. Additional research questions were added after performing two pilot interviews (in the next phase, see Section 3.2). In this paper, the following research questions are addressed in the context of software development:

RQ1 What are the current challenges, or issues, in achieving REVV alignment?

RQ2 What are the current practices that support achieving REVV alignment?

RQ3 Which current challenges are addressed by which current practices?

The main concepts of REVV alignment to be used in this study were identified after discussions and a conceptual model of the scope of the study was defined (see Fig. 2). This model was based on a traditional V-model showing the artefacts and processes covered by the study, including the relationships between artefacts of varying abstraction level and between processes and artefacts. The discussions undertaken in defining this conceptual model led to a shared understanding within the group of researchers and reduced researcher variation, thus ensuring greater validity of the data collection and results. The model was utilised both as a guide for the researchers in subsequent phases of the study and during the interviews.

3.2 Design and Planning

In this phase, the detailed research procedures for the case study were designed and preparations were made for data collection. These preparations included designing the interview guide and selecting the cases and interviewees.

The interview guide was based on the research questions and aspects, and the conceptual model produced in the *Definition* phase (see Figs. 1 and 2). The guide was constructed and refined several times by three researchers and reviewed by another four. User scenarios related to aligning requirements and testing, and examples of alignment metrics were included in the guide as a basis for discussions with the interviewees. The interview questions were mapped to the research questions to ensure that they were all covered. The guide was updated twice; after two pilot interviews, and after six initial interviews. Through these iterations the general content of the guide remained the same, though the structure and order of the interview questions were modified and improved. The resulting interview guide is published by Runeson *et al.* (2012, appendix C). Furthermore, a consent information letter was prepared to make each interviewee aware of the conditions of the interviews and their rights to refuse to answer and to withdraw at any time. The consent letter is published by Runeson *et al.* [413, Appendix E].

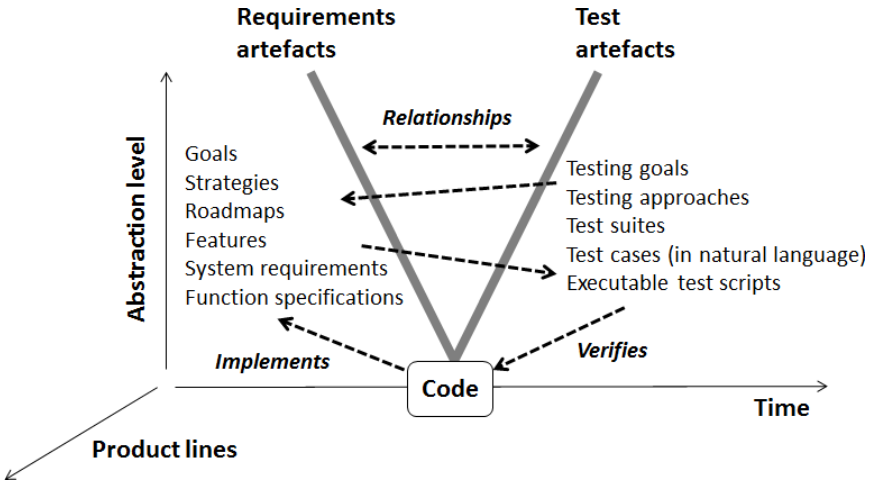


Figure 2: The conceptual model of the area under study, produced in phase 1.

The case selection was performed through a brainstorming session held within the group of researchers where companies and interviewee profiles that would match the research goals were discussed. In order to maximise the variation of companies selected from the industrial collaboration network, with respect to size, type of process, application domain and type of product, a combination of maximum variation selection and convenience selection was applied [413, pp.35 and pp.112]. The characteristics of the case companies are briefly summarised in Table 1. It is clear from the summary that they represent: a wide range of domains; size from 50 to 1,000 software developers; bespoke and market driven development; waterfall and iterative processes; using open source components or not, etc. At the time of the interviews a major shift in process model, from waterfall to agile, was underway at Company F. Hence, for some affected factors in Table 1, information is given as to for which model the data is valid.

Our aim was to cover processes and artefacts relevant to REVV alignment for the whole life cycle from requirements definition through development to system testing and maintenance. For this reason, interviewees were selected to represent the relevant range of viewpoints from requirements to testing, both at managerial and at engineering level. Initially, the company contact persons helped us find suitable people to interview. This was complemented by *snowball sampling* [407] by asking the interviewees if they could recommend a person or a role in the company whom we could interview in order to get alignment-related information. These suggestions were then matched against our aim to select interviewees in order to obtain a wide coverage of the processes and artefacts of interest. The selected interviewees represent a variety of roles, working with requirements, testing and development; both engineers and managers were interviewed. The number of in-

terviews per company was selected to allow for going in-depth in one company (Company F) through a large number of interviews. Additionally, for this large company the aim was to capture a wide view of the situation and thus mitigate the risk of a skewed sampled. For the other companies, three interviews were held per company. An overview of the interviewees, their roles and level of experience is given in Table 2. Note that for Company B, the consultants that were interviewed typically take on a multitude of roles within a project even though they can mainly be characterised as software developers they also take part in requirements analysis and specification, design and testing activities.

3.3 Evidence Collection

A semi-structured interview strategy [407] was used for the interviews, which were performed over a period of 1 year starting in May 2009. The interview guide [413, Appendix C] acted as a checklist to ensure that all selected topics were covered. Interviews lasted for about 90 min. Two or three researchers were present at each interview, except for five interviews, which were performed by only one researcher. One of the interviewers led the interview, while the others took notes and asked additional questions for completeness or clarification. After consent was given by the interviewee audio recordings were made of each interview. All interviewees consented.

The audio recordings were transcribed word by word and the transcriptions were validated in two steps to eliminate un-clarities and misunderstandings. These steps were: (i) another researcher, primarily one who was present at the interview, reviewed the transcript, and (ii) the transcript was sent to the interviewee with sections for clarification highlighted and the interviewee had a chance to edit the transcript to correct errors or explain what they meant. These modifications were included into the final version of the transcript, which was used for further data analysis.

The transcripts were divided into chunks of text consisting of a couple of sentences each to enable referencing specific parts of the interviews. Furthermore, an anonymous code was assigned to each interview and the names of the interviewees were removed from the transcripts before data analysis in order to ensure anonymity of the interviewees.

3.4 Data Analysis

Once the data was collected through the interviews and transcribed (see Fig. 1), a three-stage analysis process was performed consisting of: coding, abstraction and grouping, and interpretation. These multiple steps were required to enable the researchers to efficiently navigate and consistently interpret the huge amounts of qualitative data collected, comprising more than 300 pages of interview transcripts.

Table 1: Overview of the companies covered by this case study. At Company F a major process change was taking place at the time of the study and data specific to the previous waterfall-based process are marked with 'previous'.

Company	A	B	C	D	E	F
Type of company	Software development, embedded products	Consulting	Software development	Systems engineering, embedded products	Software development, embedded products	Software development, embedded products
# employees in software development of targeted organisation	125–150	135	500	50–100	300–350	1,000
# employees in typical project	10	Mostly 4–10, but varies greatly	50–80	software developers: 10–20	6–7 per team, 10–15 teams	Previous process: 800–1,000 person years
Distributed	No	Collocated (per project, often on-site at customer)	Yes	Yes	Yes	Yes
Domain / System type	Computer networking equipment	Advisory/technical services, application management	Rail traffic management	Automotive	Telecom	Telecom
Source of requirements	Market driven	Bespoke	Bespoke	Bespoke	Bespoke and market driven	Bespoke and market driven
Main quality focus	Availability, performance, security	Depends on customer focus	Safety	Safety	Availability, Performance, reliability, security	Performance, stability
Certification	No software related certification	No	ISO9001, ISO14001, OHSAS18001	ISO9001, ISO14001	ISO9001, ISO14001 (aiming towards adhering to TL9000)	ISO9001
Process Model	Iterative	Agile in variants	Waterfall	RUP, Scrum	Scrum, eRUP, a sprints is 3 months	Iterative with gate decisions (agile influenced). Previous: Waterfall
Duration of a typical project	6–18 months	No typical project	1–5 years to first delivery, then new software release for 1–10 years	1–5 years to first delivery, then new software releases for 1–10 years	1 year	Previous process 2 years
# requirements in typical project	100 (20–30 pages HTML)	No typical project	600–800 at system level	For software: 20–40 use cases	500–700 user stories	Previous process:14,000
# test cases in a typical project	~1,000 test cases	No typical project	250 at system level		11,000+	Previous process 200,000 at platform level, 7,000 at system level
Product Lines	Yes	No	Yes	Yes	Yes	Yes
Open Source	Yes	Yes, Wide use, including contributions	Yes, partly	No	No	Yes (with new agile process model)

Table 2: Overview of interviewees' roles at their companies incl. level of experience in that role; S(enior) = more than 3 years, or J(unior) = up to 3 years. Xn refers to interviewee n at Company X. Note: most interviewees have additional previous experience.

Role	**A**	**B**	**C**	**D**	**E**	**F**
Reqts. engineer						F1(S), F6(S), F7(S)
Systems architect				D3(J)	E1(S)	F4(S)
Software developer		B1(J), B2(S), B3(S)				
Test engineer	A2(S)		C1(S), C2(J)	D2(S)	E3(S)	F9(S), F10(S), F11(J), F12(S), F14(S)
Project manager	A1(J)		C3(S)	D1(S)		F3(J), F8(S)
Product manager	A3(S)				E2(S)	
Process manager						F2(J), F5(S), F15(J)

Coding of the transcripts, i.e. the chunks, was performed to enable locating relevant parts of the large amounts of interview data during analysis. A set of codes, or keywords, based on the research and interview questions was produced, initially at a workshop with the participating researchers. This set was then iteratively updated after exploratory coding and further discussions. In the final version, the codes were grouped into multiple categories at different abstraction levels, and a coding guide was developed. To validate that the researchers performed coding in a uniform way, one interview transcript was selected and coded by all researchers. The differences in coding were then discussed at a workshop and the coding guide was subsequently improved. The final set of codes was applied to all the transcripts. The coding guide and some coding examples are published by Runeson *et al.* [413, Appendix D].

Abstraction and grouping of the collected data into statements relevant to the goals and questions for our study was performed in order to obtain a manageable set of data that could more easily be navigated and analysed. The statements can be seen as an index, or common categorisation of sections belonging together, in essence a summary of them as done by Gorschek and Wohlin [199, 200], Pettersson *et al.* [375] and Höst *et al.* [224]. The statements were each given a unique identifier, title and description. Their relationship to other statements, as derived from the transcripts, was also abstracted. The statements and relationships between them were represented by nodes connected by directional edges. Figure 3 shows an example of the representation designed and used for this study. In particular, the figure shows the abstraction of the interview data around cross-role reviews of requirements, represented by node N4. For example, the statement 'cross-role reviews' was found to contribute to statements related to requirements quality. Each statement is represented by a node. For example, N4 for 'cross-role review', and N1, N196 and N275 for the statements related to requirements quality. The connections between these statements are represented by a 'contributes to' relationship from N4 to each of N1, N196 and N275. These connections are denoted by a directional edge tagged with the type of relationship. For example, the tags 'C' for 'contributes to', 'P' for 'prerequisite for' and 'DC' for 'does not contribute to'. In addition, negation of one or both of the statements can be denoted by applying a post- or prefix 'not' (N) to the connection. The type of relationships used for modelling the connections between statements were discussed, defined and agreed on in a series of work meetings. Traceability to the origin of the statements and the relationships between them was captured and maintained by noting the id of the relevant source chunk, both for nodes and for edges. This is not shown in Figure 3.

The identified statements including relationships to other statements were extracted per transcript by one researcher per interview. To ensure a consistent abstraction among the group of researchers and to enhance completeness and correctness, the abstraction for each interview was reviewed by at least one other researcher and agreed after discussing differences of opinion. The nodes and edges

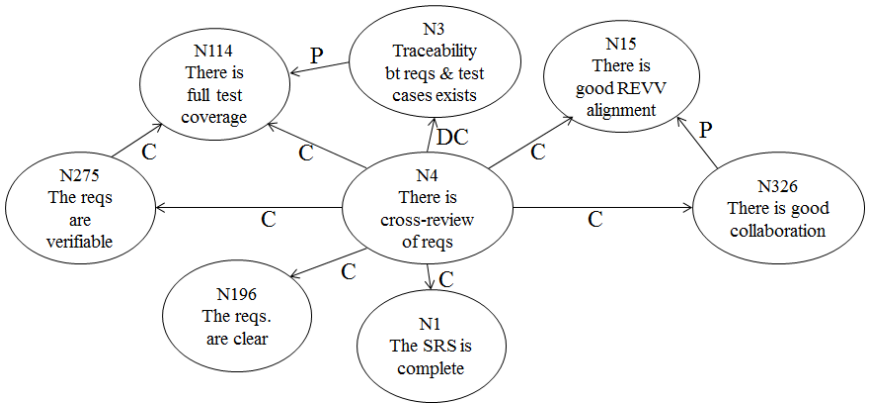


Figure 3: Part of the abstraction representing the interpretation of the interviewee data. The relationships shown denote C - contribute to, P - prerequisite for, and DC - does not contribute to.

identified by each researcher were merged into one common graph consisting of 341 nodes and 552 edges.

Interpretation of the collected evidence involved identifying the parts of the data relevant to a specific research question. The abstracted statements derived in the previous step acted as an index into the interview data and allowed the researchers to identify statements relevant to the research questions of challenges and practices. This interpretation of the interview data was performed by analysing a graphical representation of the abstracted statements including the connections between them. Through the analysis nodes and clusters of nodes related to the research questions were identified. This is similar to explorative coding and, for this paper, the identified codes or clusters represented REVV alignment challenges and practices with one cluster (code) per challenge and per practice. Due to the large amount of data, the analysis and clustering was initially performed on subsets of the graphical representation, one for each company. The identified clusters were then iteratively merged into a common set of clusters for the interviews for all companies. For example, for the nodes shown in Figure 3 the statements 'The requirements are clear' (N196) and 'The requirements are verifiable' (N275) were clustered together into the challenge 'Defining clear and verifiable requirements' (challenge Ch3.2, see Section 4.1) based on connections (not shown in the example) to other statements reflecting that this leads to weak alignment.

Even with the abstracted representation of the interview transcripts, the interpretation step is a non-trivial task which requires careful and skilful consideration to identify the nodes relevant to specific research questions. For this reason, the clustering that was performed by Bjarnason was reviewed and agreed with Runeson. Furthermore, the remaining un-clustered nodes were reviewed by Engström,

and either mapped to existing clusters, suggested for new clusters or judged to be out of scope for the specific research questions. This mapping was then reviewed and agreed with Bjarnason.

Finally, the agreed clusters were used as an index to locate the relevant parts of the interview transcripts (through traces from the nodes and edges of each cluster to the chunks of text). For each identified challenge and practice, and mapping between them, the located parts of the transcriptions were then analysed and interpreted, and reported in this paper in Sections 4.1, 4.2 and 4.3, respectively for challenges, practices, and the mapping.

3.5 Threats to Validity

There are limitations and threats to the validity to all empirical studies, and so also for this case study. As suggested by Runeson and Höst [412, 413], the construct validity, external validity and reliability were analysed in the phases leading up to the *analysis* phase of the case study, see Figure 1. We also report measures taken to improve the validity of the study.

Construct Validity

Construct validity refers to how well the chosen research method has captured the concepts under study. There is a risk that academic researchers and industry practitioners may use different terms and have different frames of reference, both between and within these categories of people. In addition, the presence of researchers may threaten the interviewees and lead them to respond according to assumed expectations. The selection of interviewees may also give a limited or unbalanced view of the construct. In order to mitigate these risks, we took the following actions in the design step:

- *Design of the interview guide and reference model:* The interview guide was designed based on the research questions and reviewed for completeness and consistency by other researchers. It was piloted during two interviews and then revised again after another six. The risk that the language and terms used may not be uniformly understood was addressed by producing a conceptual model (see Fig. 2), which was shown to the interviewees to explain the terminology. However, due to the semi-structured nature of the guide and the different interviewers involved the absence of interviewee data for a certain concept, challenge or practice cannot be interpreted as the absence of this item either in the interviewees experience or in the company. For similar reasons, the results do not include any ranking or prioritisation as to which challenges and practices are the most frequent or most effective.

- *Prolonged involvement:* The companies were selected so that at least one of the researchers had a long-term relation with them. This relationship helped

provide the trust needed for openness and honesty in the interviews. To mitigate the bias of knowing the company too well, all but five interviews (Companies D and E) were conducted by more than one interviewer.

- *Selection of interviewees:* To obtain a good representation of different aspects, a range of roles were selected to cover requirement, development and testing, and also engineers as well as managers, as reported in Table 2. The aim was to cover the relevant aspects described in the conceptual model, produced during the *Definition* phase (see Section 3.1, Figs. 1 and 2). There is a risk that the results might be biased due to a majority of the interviewees being from Company F. However, the results indicate that this risk was minor, since a majority of the identified items (see Section 4) could be connected to multiple companies.

- *Reactive bias:* The presence of a researcher might limit or influence the outcome either by hiding facts or responding after assumed expectations. To reduce this threat the interviewees were guaranteed anonymity both within the company and externally. In addition, they were not given any rewards for their participation and had the right to withdraw at any time without requiring an explanation, though no interviewees did withdraw. This approach indicated that we were interested in obtaining a true image of their reality and encouraged the interviewees to share this.

Internal Validity

Even though the conclusions in this paper are not primarily about causal relations, the identification of challenges and practices somewhat resembles identifying factors in casual relations. In order to mitigate the risk of identifying incorrect factors, we used data source triangulation by interviewing multiple roles at a company. Furthermore, extensive observer triangulation was applied in the analysis by always including more than one researcher in each step. This strategy also partly addressed the risk of incorrect generalisations when abstracting challenges and practices for the whole set of companies. However, the presented results represent one possible categorisation of the identified challenges and practices. This is partly illustrated by the fact that not all identified practices can be connected to a challenge.

The interviews at one of the case companies were complicated by a major process change that was underway at the time of the study. This change posed a risk of confusing the context for which a statement had been experienced; the previous (old) way of working or the newly introduced agile practices. To mitigate this risk, we ensured that we correctly understood which process the response concerned, i.e. the previous or the current process.

Furthermore, due to the nature of semi-structured interviews in combination with several different interviewers it is likely that different follow-on questions

were explored by the various researchers. This risk was partly mitigated by jointly defining the conceptual model and agreeing on a common interview guide that was used for all interviews. However, the fact remains that there are differences in the detailed avenues of questioning which has resulted in only being able to draw conclusions concerning what was actually said at the interviews. So, for example, if the completeness of the requirements specification (Ch3.2) was not explicitly discussed at an interview no conclusions can be drawn concerning if this is a challenge or not for that specific case.

External Validity

For a qualitative study like this, external validity can never be assured by sampling logic and statistical generalisation, but by analytical generalisation which enables drawing conclusions and, under certain conditions, relating them also to other cases [407,413]. This implies that the context of the study must be compared to the context of interest for the findings to be generalised to. To enable this process, we report the characteristics of the companies in as much detail as possible considering confidentiality (see Table 1). The fact that six different companies of varying size and domain are covered by the study, and some results are connected to the variations between them indicates that the results are more general than if only one company had been studied. But, of course, the world consists of more than six kinds of companies, and any application of the results of this study need to be mindfully tailored to other contexts.

Reliability

The reliability of the study relates to whether the same outcome could be expected with another set of researchers. For qualitative data and analysis, which are less procedural than quantitative methods, exact replication is not probable. The analysis lies in interpretation and coding of words, and the set of codes would probably be partly different with a different set of researchers.

To increase the reliability of this study and to reduce the influence by single researchers, several researchers have taken part in the study in different roles. All findings and each step of analysis have been reviewed by and agreed with at least one other researcher. In addition, a systematic and documented research process has been applied (see Fig. 1) and a trace of evidence has been retained for each analysis steps. The traceability back to each source of evidence is documented and kept even in this report to enable external assessment of the chain of evidence, if confidentially agreements would allow.

Finally, the presentation of the findings could vary depending on categorisation of the items partly due to variation in views and experience of individual researchers. For example, a challenge in achieving alignment such as Ch2 *Collaborating successfully* (see Section 4.1) could be identified also as a practice at the general level, e.g. *to collaborate successfully* could be defined as an alignment

practice. However, we have chosen to report specific practices that may improve collaboration and thereby REVV alignment. For example, P1.1 *Customer communication at all requirements levels and phases* can support improved coordination of requirements between the customer and the development team. To reduce the risk of bias in this aspect, the results and the categorisation of them was first proposed by one researcher and then reviewed by four other researchers leading to modifications and adjustments.

4 Results

Practitioners from all six companies in the study found alignment of RE with VV to be an important, but challenging, factor in developing products. REVV alignment was seen to affect the whole project life cycle, from the contact with the customer and throughout software development. The interviewees stated clearly that good alignment is essential to enable smooth and efficient software development. It was also seen as an important contributing factor in producing software that meets the needs and expectations of the customers. A software developer stated that alignment is 'very important in creating the right system' (B1:27[1]). One interviewee described the customer's view of a product developed with misaligned requirements as: 'There wasn't a bug, but the behaviour of the functionality was interpreted or implemented in such a way that it was hard to do what the customer [originally] intended' (A3:43). Another interviewee mentioned that alignment between requirements and verification builds customer trust in the end product since good alignment allows the company to 'look into the customer's eyes and explain what have we tested... on which requirements' (D2:10).

In general, the interviewees expressed that weak and unaligned communication of the requirements often cause inconsistencies that affect the verification effort. A common view was that these inconsistencies, caused by requirements that are misunderstood, incorrect or changed, or even un-communicated, leads to additional work in updating and re-executing test cases. Improved alignment, on the other hand, was seen to make 'communication between different levels in the V-model a lot easier' (E3:93). One of the interviewed testers stated: 'Alignment is necessary. Without it we [testers] couldn't do our job at all' (C1:77).

Below, we present the results concerning the challenges of alignment (Ch1-Ch10) and the practices (P1-P10) used, or suggested, by the case companies to address REVV challenges. Table 3 provides an overview of the challenges found for each company, while Figure 4 contains an overview of the practices. Figure 6 shows which challenges each practices is seen to address.

[1] Reference to source is given by interviewee code, see Table 2.

Table 3: Alignment challenges mentioned for each company. Note: a blank cell means that the challenge was not mentioned during the interviews, not that it is not experienced.

	Id	Challenge	Company					
			A	B	C	D	E	F
	Ch1	Aligning goals and perspectives within an organisation	X	X	X		X	X
	Ch2	Cooperating successfully	X		X	X	X	X
Req spec quality	Ch3.1	Defining clear and verifiable requirements			X	X	X	X
	Ch3.2	Defining complete requirements		X		X	X	X
	Ch3.3	Keeping requirements documents updated						X
VV quality	Ch4.1	Full test coverage	X	X	X	X		X
	Ch4.2	Defining a good verification process						X
	Ch4.3	Verifying quality requirements		X		X		X
	Ch5	Maintaining alignment when requirements change	X		X			X
Req's abstract levels	Ch6.1	Defining requirements at abstraction level well matched to test cases				X		X
	Ch6.2	Coordinating requirements at different abstraction levels	X					X
Traceability	Ch7.1	Tracing between requirements and test cases	X	X	X	X		X
	Ch7.2	Tracing between requirements abstraction levels		X	X	X		
	Ch8	Time and resource availability			X		X	X
	Ch9	Managing a large document space			X	X		X
	Ch10	Outsourcing of components or testing				X		X

4.1 Alignment Challenges

The alignment challenges identified through this study are summarised in Table 3. Some items have been categorised together as one challenge, resulting in 10 main challenges where some consist of several related challenges. For example, Ch3 *Requirements specification quality* consists of three challenges (Ch3.1-Ch3.3) concerning different aspects of requirements quality. Each challenge including sub items is described in the subsections that follow.

Challenge 1: Aligning Goals and Perspectives within an Organisation (Ch1)

The alignment of *goals* throughout the organisation was mentioned by many interviewees as vital in enabling cooperation among different organisational units (see Ch2 in Section 4.1). However, goals were often felt to be missing or unclearly defined, which could result in 'making it difficult to test [the goals]' (B3:17). In several companies problems with differing and unaligned goals were seen to affect the synchronisation between requirements and testing, and cause organisational units to counteract each other in joint development projects. For example, a product manager mentioned that at times, requirement changes needed from a business

perspective conflicted with the goals of the development units; 'They [business roles] have their own directives and... schedule target goals' and 'they can look back and see which product was late and which product was good' (A3:74). In other words, misaligned goals may have an impact on both time schedules and product quality.

Many interviewees described how awareness and understanding of different *perspectives* on the problem domain is connected to better communication and cooperation, both towards the customers and external suppliers, and internally between competence areas and units. When there is a lack of aligned perspectives, the customer and the supplier often do not have the same understanding of the requirements. This may result in 'errors in misunderstanding the requirements' (B3:70). Lack of insight into and awareness of different perspectives was also seen to result in decisions (often made by other units) being questioned and requirements changed at a late stage in the development cycle with a subsequent increase in cost and risk. For example, a systems architect described that in a project where there is a 'higher expectations on the product than we [systems architect] scoped into it' (E1:20) a lot of issues and change requests surface in the late project phases. A software developer stated concerning the communication between requirements engineers and developers that 'if both have a common perspective [of technical possibilities], then it would be easier to understand what [requirements] can be set and what cannot be set' (F13:29). Or in other words, with an increased common understanding technically infeasible requirements can be avoided already at an early stage.

Weak alignment of goals and perspectives implies a weak coordination at higher organisational levels and that strategies and processes are not synchronised. As stated by a process manager, the involvement of many separate parts of an organisation then leads to 'misunderstandings and misconceptions and the use of different vocabulary' (F2:57). In addition, a test engineer at Company A mentioned that for the higher abstraction levels there were no attempts to synchronise, for example, the testing strategy with the goals of development projects to agree on important areas to focus on (A2:105). Low maturity of the organisation was thought to contribute to this and result in the final product having a low degree of correspondence to the high-level project goals. A test engineer said: 'In the long run, we would like to get to the point where this [product requirements level] is aligned with this [testing activities]' (A2:119).

Challenge 2: Cooperating Successfully (Ch2)

All of the companies included in our study described close cooperation between roles and organisational units as vital for good alignment and coordination of both people and artefacts. Weak cooperation is experienced to negatively affect the alignment, in particular at the product level. A product manager stated that 'an "us and them" validation of product level requirements is a big problem' (A3:58-59).

Ensuring clear agreement and communication concerning which requirements to support is an important collaboration aspect for the validation. At Company F (F12:63) lack of cooperation in the early phases in validating requirements has been experienced to result in late discovery of failures in meeting important product requirements. The development project then say at a late stage: 'We did not approve these requirements, we can't solve it' (F12:63) with the consequence that the requirements analysis has to be re-done. For Company B (consulting in different organisations) cooperation and communication was even described as being prioritised above formal documentation and processes, expressed as: 'We have succeeded with mapping requirements to tests since our process is more of a discussion' (B3:49). Several interviewees described that alignment at product and system level, in particular, is affected by how well people cooperate (C2:17, E1:44, 48, E2:48, F4:66, F15:46). When testers have a good cooperation and frequently communicate with both requirements-related and development-related roles, this leads to increased alignment (E3:93).

Organisational boundaries were mentioned as further complicating and hindering cooperation between people for two of the companies, namely Companies E and F. In these cases, separate organisational units exist for requirements (E2:29, E3:94, F2:119), usability (F10:108) and testing (F3:184). As one interviewee said: 'it is totally different organisations, which results in. . . misunderstandings and misconceptions. . . we use different words' (F2:57). Low awareness of the responsibilities and tasks of different organisational units was also claimed to negatively affect alignment (F2:264). This may result in increased lead times (E1:44, F15:33), need for additional rework (E1:150, E1:152), and conflicts in resource allocation between projects (F10:109, E1:34).

Challenge 3: Good Requirements Specification Quality (Ch3)

'If we don't have good requirements the tests will not be that good' (D3:14). When the requirement specification is lacking the testers need to guess and make up the missing information since 'the requirements are not enough for writing the software and testing the software' (D3:19). This both increases the effort required for testing and the risk of misinterpretation and missing vital customer requirements. One process manager expressed that the testability of requirements can be improved by involving testers and that 'one main benefit [of alignment] is improving the requirements specifications' (F2:62). A test leader at the same company identified that a well aligned requirements specification (through clear agreement between roles and tracing between artefacts) had positive effects such as 'it was very easy to report when we found defects, and there were not a lot of discussions between testers and developers, because everyone knew what was expected' (F9:11).

There are several aspects to good requirements that were found to relate to alignment. In the study, practitioners mentioned good requirements as being veri-

fiable, clear, complete, at the right level of abstraction, and up-to-date. Each aspect is addressed below.

- **Defining clear and verifiable requirements (Ch3.1)** was mentioned as a major challenge in enabling good alignment of requirements and testing, both at product and at detailed level. This was mentioned for four of the six companies covered by our study, see Table 3. Unclear and non-verifiable requirements were seen as resulting in increased lead times and additional work in later phases in clarifying and redoing work based on unclear requirements (F2:64, D1:80). One test manager said that 'in the beginning the requirements are very fuzzy. So it takes time. And sometimes they are not happy with our implementation, and we have to do it again and iterate until it's ready' (F11:27, similar in E3:44). Failure to address this challenge ultimately results in failure to meet the customer expectations with the final product. A project manager from Company D expressed this by saying that non-verifiable requirements is the reason 'why so many companies, developers and teams have problems with developing customer-correct software' (D1:36).

- **Defining complete requirements (Ch3.2)** was claimed to be required for successful alignment by interviewees from four companies, namely Companies B, D, E and F. As expressed by a systems architect from Company D, 'the problem for us right now is not [alignment] between requirements and testing, but that the requirements are not correct and complete all the time' (D3:118). Complete requirements support achieving full test coverage to ensure that the full functionality and quality aspects are verified (F14:31). When testers are required to work with incomplete requirements, additional information is acquired from other sources, which requires additional time and effort to locate (D3:19).

- **Keeping requirements documentation updated (Ch3.3)** Several interviewees from Company F described how a high frequency of change leads to the requirements documentation not being kept updated, and consequently the documentation cannot be relied on (F14:44, F5:88). When a test for a requirement then fails, the first reaction is not: 'this is an error', but rather 'is this really a relevant requirement or should we change it' (F5:81). Mentioned consequences of this include additional work to locate and agree to the correct version of requirements and rework (F3:168) when incorrect requirements have been used for testing. Two sources of requirements changes were mentioned, namely requested changes that are formally approved (F14:50), but also changes that occur as the development process progresses (during design, development etc.) that are not raised as formal change requests (F5:82, F5:91, F11:38). When the requirements documentation is not reliable, the projects depend on individuals for correct requirements information. As expressed by one requirements engineer: 'when you

lose the people who have been involved, it is tough. And, things then take more time' (F1:137).

Challenge 4: Validation and Verification Quality (Ch4)

Several issues with validation and verification were mentioned as alignment challenges that affect the efficiency and effectiveness of the testing effort. One process manager with long experience as a tester said: 'We can run 100,000 test cases but only 9% of them are relevant' (F15:152). Testing issues mentioned as affecting alignment were: obtaining full test coverage, having a formally defined verification process and the verification of quality requirements.

- **Full test coverage (Ch4.1)** Several interviewees described full test coverage of the requirements as an important aspect of ensuring that the final product fulfils the requirements and the expectations of the customers. As one software developer said: 'having full test coverage with unit tests gives a better security... check that I have interpreted things correctly with acceptance tests' (B1:117). However, as a project manager from Company C said: 'it is very hard to test everything, to think about all the complexities' (C3:15). *Unclear* (Ch3.2, C1:4) and *non-verifiable requirements* (Ch3.1, A1:55, D1:78, E1:65) were mentioned as contributing to difficulties in achieving full test coverage of requirements for Companies A, B, D and E. For certain requirements that are expressed in a verifiable way a project manager mentioned that they cannot be tested due to limitations in the process, competence and test tools and environments (A1:56). To ensure full test coverage of requirements the testers need knowledge of the full set of requirements, which is impeded in the case of *incomplete requirements specifications* (Ch3.3) where features and functionality are not described (D3:16). This can also be the case for *requirements defined at a higher abstraction level* (F2:211, F14:56). *Lack of traceability* between requirements and test cases was stated to making it harder to know when full *test coverage* has been obtained (A1:42). For Company C, traceability was stated as time consuming but necessary to ensure and demonstrate full test coverage, which is mandatory when producing safety-critical software (C1:6, C1:31). Furthermore, obtaining sufficient coverage of the requirements requires analysis of both the requirement and the connected test cases (C1:52, D3:84, F14:212). As one requirements engineer said, 'a test case may cover part of a requirement, but not test the whole requirement' (F7:52). Late requirements changes was mentioned as a factor contributing to the challenge of full test coverage (C1:54, F7:51) due to the need to update the affected test cases, which is hampered by failure to keep the requirements specification updated after changes (Ch3.5, A2:72, F15:152).

- **Having a verification process (Ch4.2)** was mentioned as directly connected to good alignment between requirements and test. At Company F, the ongoing shift towards a more agile development process had resulted in the verification unit operating without a formal process (F15:21). Instead each department and project 'tries to work their own way... that turns out to not be so efficient' (F15:23), especially so in this large organisation where many different units and roles are involved from the initial requirements definition to the final verification and launch. Furthermore, one interviewee who was responsible for defining the new verification process (F15) said that 'the hardest thing [with defining a process] is that there are so many managers... [that don't] know what happens one level down'. In other words, a verification process that supports requirements-test alignment needs to be agreed with the whole organisation and at all levels.

- **Verifying quality requirements (Ch4.3)** was mentioned as a challenge for Companies B, D and F. Company B has verification of quality in focus with continuous monitoring of quality levels in combination with frequent releases; 'it is easy to prioritise performance optimisation in the next production release' (B1:52). However, they do not work pro-actively with quality requirements. Even though they have (undocumented) high-level quality goals the testers are not asked to use them (B1:57, B2:98); 'when it's not a broken-down [quality] requirement, then it's not a focus for us [test and development]' (B3:47). Company F does define formal quality requirements, but these are often not fully agreed with development (F12:61). Instead, when the specified quality levels are not reached, the requirements, rather than the implementation, are changed to match the current behaviour, thus resigning from improving quality levels in the software. As one test engineer said: 'We currently have 22 requirements, and they always fail, but we can't fix it' (F12:61).

  Furthermore, defining verifiable quality requirements and test cases was mentioned as challenging, especially for usability requirements (D3:84, F10:119). Verification is then faced with the challenge of subjectively judging if a requirement is passed or failed (F2:46, F10:119). At Company F, the new agile practices of detailing requirements at the development level together with testers was believed to, at least partly, address this challenge (F12:65). Furthermore, additional complication is that some quality requirements can only be verified through analysis and not through functional tests (D3:84).

Challenge 5: Maintaining Alignment when Requirements Change (Ch5)

Most of the companies of our study face the challenge of maintaining alignment between requirements and tests as requirements change. This entails ensuring that both artefacts and tracing between them are updated in a consistent manner. Company B noted that the impact of changes is specifically challenging for test since

test code is more sensitive to changes than requirements specifications. 'That's clearly a challenge, because [the test code is] rigid, as you are exemplifying things in more detail. If you change something fundamental, there are many tests and requirements that need to be modified' (B3:72).

Loss of traces from test cases to requirements over time was also mentioned to cause problems. When test cases for which traces have been outdated or lost are questioned, then 'we have no validity to refer to... so we have to investigate' (A2:53). In Company A, the connection between requirements and test cases are set up for each project (A2:71): 'This is a document that dies with the project'; a practice found very inefficient. Other companies had varying ambitions of a continuous maintenance of alignment and traces between the artefacts. A key for maintaining alignment when requirements change is that the requirements are actively used. When this is not the case there is a need for obtaining requirements information from other sources. This imposes a risk that 'a requirement may have changed, but the software developers are not aware of it' (D3:97).

Interviewees implicitly connected the traceability challenge to tools, although admitting that 'a tool does not solve everything... Somebody has to be responsible for maintaining it and to check all the links... if the requirements change' (C3:53). With or without feasible tools, tracing also requires personal assistance. One test engineer said, 'I go and talk to him and he points me towards somebody' (A2:195).

Furthermore, the *frequency of changes* greatly affects the extent of this challenge and is an issue when trying to establish a base-lined version of the requirements. Company C has good tool support and traceability links, but require defined versions to relate changes to. In addition, they have a product line, which implies that the changes must also be coordinated between the platform (product line) and the applications (products) (C3:19, C3:39).

Challenge 6: Requirements Abstraction Levels (Ch6)

REVV alignment was described to be affected by the abstraction levels of the requirements for Companies A, D and F. This includes the relationship to the abstraction levels of the test artefacts and ensuring consistency between requirements at different abstraction levels.

- **Defining requirements at abstraction levels well-matched to test cases (Ch6.1)** supports defining test cases in line with the requirements and with a good coverage of them. This was mentioned for Companies D and F. A specific case of this at Company D is when the testers 'don't want to test the complete electronics and software system, but only one piece of the software' (D3:56). Since the requirements are specified at a higher abstraction level than the individual components, the requirements for this level then need to be identified elsewhere. Sources for information mentioned by the interviewees include the design specification, asking people or making up the missing requirements (D3:14). This is also an issue when retesting only

parts of a system which are described by a high-level requirement to which many other test cases are also traced (D3:56). Furthermore, synchronising the abstraction levels between requirements and test artefacts was mentioned to enhance coverage (F14:31).

- **Coordinating requirements at different abstraction levels (Ch6.2)** when breaking down the high-level requirements (such as goals and product concepts) into detailed requirements at system or component level was mentioned as a challenge by several companies. A product manager described that failure to coordinate the detailed requirements with the overall concepts could result in that 'the intention that we wanted to fulfil is not solved even though all the requirements are delivered' (A3:39). On the other hand, interviewees also described that the high-level requirements were often vague at the beginning when 'it is very difficult to see the whole picture' (F12:144) and that some features are 'too complex to get everything right from the beginning' (A3:177).

Challenge 7: Tracing Between Artefacts (Ch7)

This challenge covers the difficulties involved in tracing requirements to test cases, and vice versa, as well as, tracing between requirements at different abstraction levels. Specific tracing practices identified through our study are described in Section 4.2.

- **Tracing between requirements and test cases (Ch7.1)**. The most basic kind of traceability, referred to as 'conceptual mapping' in Company A (A2:102), is having a line of thought (not necessarily documented) from the requirements through to the defining and assessing of the test cases. This cannot be taken for granted. Lack of this basic level of tracing is largely due to weak awareness of the role requirements in the development process. As a requirements process engineer in Company F says, 'One challenge is to get people to understand why requirements are important; to actually work with requirements, and not just go off and develop and do test cases which people usually like doing' (F5:13).

  Tracing by using matrices to map between requirements and test cases is a major cost issue. A test architect at Company F states, that 'we don't want to do that one to one mapping all the way because that takes a lot of time and resources' (F10:258). Companies with customer or market demands on traceability, e.g. for safety critical systems (Companies C and D), have full traceability in place though 'there is a lot of administration in that, but it has to be done' (C1:6). However, for the other case companies in our study (B3:18, D3:45; E2:83; F1:57), it is a challenge to implement and maintain this support even though tracing is generally seen as supporting alignment. Company A says 'in reality we don't have the connections' (A2:102) and

for Company F 'in most cases there is no connection between test cases and requirements' (F1:157). Furthermore, introducing traceability may be costly due to large legacies (F1:57) and maintaining traceability is costly. However, there is also a cost for lack of traceability. This was stated by a test engineer in Company F who commented on degrading traceability practices with 'it was harder to find a requirement. And if you can't find a requirement, sometimes we end up in a phase where we start guessing' (F12:112).

Company E has previously had a tradition of 'high requirements on the traceability on the products backwards to the requirements' (E2:83). However, this company foresees problems with the traceability when transitioning towards agile working practices, and using user stories instead of traditional requirements. A similar situation is described for Company F, where they attempt to solve this issue by making the test cases and requirements one; 'in the new [agile] way of working we will have the test cases as the requirements' (F12:109).

Finally, traceability for quality (a.k.a. non-functional) requirements creates certain challenges, 'for instance, for reliability requirement you might... verify it using analysis' (D3:84) rather than testing. Consequently, there is no single test case to trace such a quality requirement to, instead verification outcome is provided through an analysis report. In addition, tracing between requirements and test cases is more difficult 'the higher you get' (B3:20). If the requirements are at a high abstraction level, it is a challenge to define and trace test cases to cover the requirements.

- **Tracing between requirements abstraction levels (Ch7.2)**. Another dimension of traceability is vertical tracing between requirements at different abstraction levels. Company C operates with a detailed requirements specification, which for some parts consists of sub-system requirements specifications (C1:31). In this case, there are no special means for vertical traceability, but pointers in the text. It is similar in Company D, where a system architect states that 'sometimes it's not done on each individual requirement but only on maybe a heading level or something like that' (D3:45). Company F use a high-end requirements management tool, which according to the requirements engineer 'can trace the requirement from top level to the lowest implementation level' (F7:50).

  Company E has requirements specifications for different target groups, and hence different content; one market oriented, one product oriented, and one with technical details (E1:104). The interviewee describes tracing as a 'synch activity' without specifying in more detail. Similarly, Company F has 'roadmaps' for the long term development strategy, and there is a loosely coupled 'connection between the roadmaps and the requirements' to balance the project scope against strategy and capacity (F11:50).

Challenge 8: Time and Resource Availability (Ch8)

In addition to the time consuming task of defining and maintaining traces (Ch7) further issues related to time and resources were brought forward in Companies C, E and F. Without sufficient resources for validation and verification the amount of testing that can be performed is not sufficient for the demands on functionality and quality levels expected of the products. The challenge of planning for *enough test resources* is related to the alignment between the proposed requirements and the time and resources required to sufficiently test them. A requirements engineer states that 'I would not imagine that those who are writing the requirements in anyway are considering the test implications or the test effort required to verify them' (F6:181). A test manager confirms this view (F14:188). It is not only a matter of the amount of resources, but also in which time frame they are available (E1:18). Furthermore, availability of all the necessary *competences and skills* within a team was also mentioned as an important aspect of ensuring alignment. A software developer phrased it: 'If we have this kind of people, we can set up a team that can do that, and then the requirements would be produced properly and hopefully 100% achievable' (F13:149). In addition, experienced individuals were stated to contribute to strengthening the alignment between requirements and testing, by being 'very good at knowing what needs to be tested and what has a lower priority' (C2:91), thereby increasing the test efficiency. In contrast, inexperienced testing teams were mentioned for Company C as contributing to weaker alignment towards the overall set of requirements including goals and strategies since they 'verify only the customer requirements, but sometimes we have hazards in the system which require the product to be tested in a better way' (C2:32-33).

Challenge 9: Managing a Large Document Space (Ch9)

The main challenge regarding the information management problems lies in the sheer numbers. A test engineer at Company F estimates that they have accumulated 50,000 requirements in their database. In addition, they have 'probably hundreds of thousands of test cases' (F2:34, F12:74). Another test engineer at the same company points out that this leads to information being *redundant* (F11:125), which consequently may lead to inconsistencies. A test engineer at Company D identifies the *constant change* of information as a challenge; they have difficulties to work against the same baseline (D2:16).

Another test engineer at Company F sees information management as a tool issue. He states that 'the requirements tool we have at the moment is not easy to work with... Even, if they find the requirements they are not sure they found the right version' (F9:81). In contrast, a test engineer at Company C is satisfied with the ability to find information in the same tool (C2). A main difference is that at Company F, 20 times as many requirements are handled than at Company C.

The investment into introducing explicit links between a huge legacy of requirements and test cases is also put forward as a major challenge for Companies

A and F. In addition, connecting and integrating different tools was also mentioned as challenging due to separate responsibilities and competences for the two areas of requirements and testing (F5:95, F5:120).

Challenge 10: Outsourcing or Offshoring of Components or Testing (Ch10)

Outsourcing and offshoring of component development and testing create challenges both in agreeing to which detailed requirements to implement and test, and in tracing between artefacts produced by different parties. Company D stresses that the ***timing*** of the outsourcing plays a role in the difficulties in tracing component requirement specifications to the internal requirements at the higher level; 'I think that's because these outsourcing deals often have to take place really early in the development' (D3:92). Company F also mentions the timing aspect for acquisition of hardware components; 'it is a rather formal structured process, with well-defined deliverables that are slotted in time' (F6:21).

When testing is outsourced, the ***specification*** of what to test is central and related to the type of testing. The set-up may vary depending on competence or cultural differences etc. For example, Company F experienced that cultural aspects influence the required level of detail in the specification; 'we [in Europe] might have three steps in our test cases, while the same test case with the same result, but produced in China, has eight steps at a more detailed level' (F15:179). A specification of what to test may be at a high level and based on a requirements specification from which the in-sourced party derives tests and executes. An alternative approach is when a detailed test specification is requested to be executed by the in-sourced party (F6:251-255).

4.2 Practices for Improved Alignment

This study has identified 27 different alignment practices, grouped into 10 categories. Most of the practices are applied at the case companies, though some are suggestions made by the interviewees. These categories and the practices are presented below and discussed and summarised in Section 5. In Section 4.3 they are mapped to the challenges that they are seen to address.

Requirements Engineering Practices

Requirements engineering practices are at the core of aligning requirements and testing. This category of practices includes customer communication and involving development-near roles in the requirements process. The interviewees described close cooperation and team work as a way to improve RE-practices (F12:146) and thereby the coordination with developers and testers and avoid a situation where product managers say "'redo it" when they see the final product' (F12:143).

Table 4: Alignment practices and categories, and case companies for which they were mentioned. Experienced practices are marked with X, while suggested practices are denoted with S.

			Company					
Cat.	Id	Description	A	B	C	D	E	F
Requirements	P1.1	Customer communication at all requirements levels and phases		X	X	X	X	X
	P1.2	Development involved in detailing requirements	X	X				X
	P1.3	Cross-role requirements reviews	X		X	X	X	X
	P1.4	Requirements review responsibilities defined					X	X
	P1.5	Subsystem expert involved in requirements definition				X		X
	P1.6	Documentation of requirement decision rationales					S	S
Validation	P2.1	Test cases reviewed against requirements						X
	P2.2	Acceptance test cases defined by customer		X				
	P2.3	Product manager reviews prototypes	X				X	
	P2.4	Management base launch decision on test report						X
	P2.5	User / Customer testing		X		X	X	X
Verification	P3.1	Early verification start					X	X
	P3.2	Independent testing			X	X	X	
	P3.3	Testers re-use customer feedback from previous projects				X	X	X
	P3.4	Training off-shore testers			X			
Change	P4.1	Process for requirements changes involving VV	X		X	X	X	X
	P4.2	Product-line requirements practices	X		X			
	P5	Process enforcement			X			S
Tracing	P6.1	Document-level traces	X					
	P6.2	Requirements-test case traces						X
	P6.3	Test cases as requirements	X					X
	P6.4	Same abstraction levels for requirements and test spec			X	X		
	P7	Traceability responsibility role			X	X	X	
Tools	P8.1	Tool support for requirements and testing	X		X	X	X	X
	P8.2	Tool support for requirements-test case tracing	X		X	X	X	X
	P9	Alignment metrics, e.g. test coverage			X	X	X	X
	P10	Job rotation				S		S

A blank cell means that the practice was not mentioned during the interviews. It does not mean that it is not applied at the company

- **Customer communication at all levels and in all phases of development (P1.1)** was mentioned as an alignment practice for all but one of the case companies. The communication may take the form of customer-supplier co-location; interaction with the customer based on executable software used for demonstrations or customer validation; or agreed acceptance criteria between customer and supplier. For the smaller companies, and especially those with bespoke requirements (Companies B and C), this interaction is directly with a physical customer. In larger companies (Companies E and F), and especially within market driven development, a *customer proxy* may be used instead of the real customer, since there is no assigned customer at the time of development or there is a large organisational distance to the customer. Company F assigns a person in each development team 'responsible for the feature scope. That person is to be available all through development and to the validation of that feature' (F2:109). Furthermore, early discussions about product roadmaps from a 4 to 5 year perspective are held with customers and key suppliers (F6:29) as an initial phase of the requirements process.

- **Involving developers and testers in detailing requirements (P1.2)** is another practice, especially mentioned by Companies A and F. A product manager has established this as a deliberate strategy by conveying the vision of the product to the engineers rather than detailed requirements: 'I'm trying to be more conceptual in my ordering, trying to say what's important and the main behaviour'(A3:51). The responsibility for detailing the specification then shifts to the development organisation. However, if there is a weak awareness of the customer or market perspectives, this may be a risky practice as 'some people will not [understand this] either because they [don't] have the background or understanding of how customers or end-users or system integrators think' (A3:47). Testers may be involved to ensure the testability of the requirements, or even specify requirements in the form of test cases. Company F was in the process of transferring from a requirements-driven organisation to a design-driven one. Splitting up the (previous) centralised requirements department resulted in 'requirements are vaguer now. So it's more up to the developers and the testers to make their own requirements' (F12:17). Close cooperation around requirements when working in an agile fashion was mentioned as vital by a product manager from Company E: 'Working agile requires that they [requirements, development, and test] are really involved [in requirements work] and not only review' (E2:83).

- **Cross-role requirements reviews (P1.3)** across requirements engineers and testers is another practice applied to ensure that requirements are understood and testable (A2:65, C3:69, F2:38, F7:7). The practical procedures for the reviews, however, tend to vary. Company A has an early review of require-

ments by testers while Companies C and D review the requirements while creating the test cases. Different interviewees from Companies E and F mentioned one or the other of these approaches; the process seems to prescribe cross-role reviews but process compliance varies. A test engineer said '[the requirements are] usually reviewed by the testers. It is what the process says' (F11:107). Most interviewees mention testers' reviews of requirements as a good practice that enhances both the communication and the quality of the requirements, thereby resulting in better alignment of the testing effort. Furthermore, this practice was described as enabling early identification of problems with the test specification avoiding (more expensive) problems later on (C2:62). A systems architect from Company F described that close collaboration between requirements and testing around quality requirements had resulted in 'one area where we have the best alignment' (F4:101).

- **Defining a requirements review responsible (P1.4)** was mentioned as a practice that ensures that requirement reviews are performed (E2:18, F2:114). In addition, for Company F this role was also mentioned as reviewing the quality of the requirements specification (F2:114) and thereby directly addressing the alignment challenge of low quality of the requirements specification (Ch3).

- **Involving domain experts in the requirements definition (P1.5)** was mentioned as a practice to achieve better synchronisation between the requirements and the system capabilities, and thereby support defining more realistic requirements. The expert 'will know if we understand [the requirement] correctly or not' (D3:38), said a system architect. Similar to the previous RE practices, this practice was also mentioned as supporting alignment by enhancing the quality of the requirements (Ch3) which are the basis for software testing.

- **Documentation of requirement decision rationales (P1.6)**, and not just the current requirement version, was suggested as a practice that might facilitate alignment by interviewees from both of the larger companies in our study, namely E and F. 'Softly communicating how we [requirements roles] were thinking' (E3:90) could enhance the synchronisation between project phases by better supporting hand-over between the different roles (F4:39). In addition, the information could support testers in analysing customer defect reports filed a long time after development was completed, and in identifying potential improvements (E3:90). However, the information needs to be easily available and connected to the relevant requirements and test cases for it to be practically useful to the testers (F1:120).

Validation Practices

Practices for validating the system under development and ensuring that it is in-line with customer expectations and that the right product is built (IEEE610, [235]) include test case reviews, automatic testing of acceptance test cases, and review of prototypes.

- **Test cases are reviewed against requirements (P2.1)** at Company F (F14:62). In their new (agile) development processes, the attributes of ISO9126 [241] are used as a checklist to ensure that not only functional requirements are addressed by the test cases, but also other quality attributes (F14:76).
- **Acceptance test cases defined by customer (P2.2)**, or by the business unit, is practiced at Company B. The communication with the customer proxy in terms of acceptance criteria for (previously agreed) user scenarios acts as a 'validation that we [software developers] have interpreted the requirements correctly' (B1:117). This practice in combination with full unit test coverage of the code (B1:117) was experienced to address the challenge of achieving full test coverage of the requirements (Ch4, see Section 4.1).
- **Reviewing prototypes (P2.3)** and GUI mock-ups was mentioned as an alignment practice applied at Company A. With this practice, the product manager in the role as customer proxy validates that the developed product is in-line with the original product intents (A3:153,163). Company partners that develop tailor-made systems using their components may also be involved in these reviews.
- **Management base launch decisions on test reports (P2.4)** was mentioned as an important improvement in the agile way of working recently introduced at Company F. Actively involving management in project decisions and, specifically in deciding if product quality is sufficient for the intended customers was seen as ensuring and strengthening the coordination between customer and business requirements, and testing; 'Management... have been moved down and [made to] sit at a level where they see what really happens' (F15:89).
- **User/customer testing (P2.5)** is a practice emphasised by Company B that apply agile development practices. At regular intervals, executable code is delivered, thus allowing the customer to test and validate the product and its progress (B3:32, B3:99). This practice is also applied at Company E, but with an organisational unit functioning as the user proxy (E3:22). For this practice to be effective the customer testing needs to be performed early on. This is illustrated by an example from Company F, namely 'before the product is launched the customer gets to test it more thoroughly. And they submit a lot of feedback. Most are defects, but there are a number of changes coming out of that. That's very late in the process... a few weeks [...] before the

product is supposed to be launched' (F1:12). If the feedback came earlier, it could be addressed, but not at this late stage.

Verification Practices

Verification ensures that a developed system is built according to the specifications (IEEE610, [235]). Practices to verify that system properties are aligned to system requirements include starting verification early to allow time for feedback and change, using independent test teams, re-use of customer feedback obtained from previous projects, and training testers at outsourced or off-shored locations.

- **Early verification (P3.1)** is put forward as an important practice especially when specialised hardware development is involved, as for an embedded product. Verification is then initially performed on prototype hardware (F15:114). Since quality requirements mostly relate to complete system characteristics, early verification of these requirements is harder, but also more important. Company E states: 'If we have performance issues or latency issues or database issues then we usually end up in weeks of debugging and checking and tuning' (E3:28).

- **Independent test teams (P3.2)** are considered a good practice to reduce bias in interpreting requirements by ensuring that testers are not influenced by the developers' interpretation of requirements. However, this practice also increases the risk of misalignment when the requirements are insufficiently communicated since there is a narrower communication channel for requirements-related information. This practice was emphasised especially for companies with safety requirements in the transportation domain (Companies C and D); 'due to the fact that this is a fail-safe system, we need to have independency between testers and designers and implementers' (C3:24, similar in C2:39, D2:80), 'otherwise they [test team] might be misled by the development team' (D1:41). Similarly, Company F emphasises alternative perspectives taken by an independent team. As a software developer said: 'You must get another point of view of the software from someone who does not know the technical things about the in-depth of the code, and try to get an overview of how it works' (F13:32).

- **Testers re-use customer feedback from previous projects (P3.3)** when planning the verification effort for later projects (F14:94), thereby increasing the test coverage. In addition to having knowledge of the market through customer feedback, verification organisations often analyse and test competitor products. With a stronger connection and coordination between the verification and business/requirements units, this information could be utilised in defining more accurate roadmaps and product plans.

- **Training off-shore/outsourced testers (P3.4)** in the company's work practices and tools increases the competence and motivation of the outsourced testers in the methods and techniques used by the outsourcing company. This was mentioned by a project manager from Company C as improving the quality of verification activities and the coordination of these activities with requirement (C3:49, C3:64).

Change Management Practices

Practices to manage the (inevitable) changes in software development may mitigate the challenge of maintaining alignment (Ch5, see Section 4.1). We identified practices related to the involvement of testing roles in the change management process and also practices connected to product lines as a means to support REVV alignment.

- **Involving testing roles in change management (P4.1)**, in the decision making and in the communication of changes, is a practice mentioned by all companies, but one, as supporting alignment through increased communication and coordination of these changes with the test organisation. '[Testers] had to show their impacts when we [product management] were deleting, adding or changing requirements' (E2:73) and 'any change in requirement... means involving developer, tester, project manager, requirements engineer; sitting together when the change is agreed, so everybody is aware and should be able to update accordingly' (F8:25). In companies with formalised waterfall processes, a change control board (CCB) is a common practice for making decisions about changes. Company D has weekly meetings of the 'change control board with the customer and we also have more internal change control boards' (D1:106). The transitioning to agile practices affected the change management process at Companies E and F. At Company F the change control board (CCB) was removed, thus enhancing local control at the expense of control of the whole development chain. As expressed by a process manager in Company F: 'I think it will be easy for developers to change it [the requirements] into what they want it to be' (F12:135). At Company E centralised decisions were retained at the CCB (E2:73), resulting in a communication challenge; 'sometimes they [test] don't even know that we [product management] have deleted requirements until they receive them [as deleted from the updated specification]' (E2:73).

- **Product-line requirements practices (P4.2) are applied** in order to reduce the impact of a requirements change. By sharing a common product line (a.k.a. platform), these companies separate between the requirements for the commonality and variability of their products. In order to reduce the impact of larger requirements changes and the risks these entail for current projects, Company A 'develop it [the new functionality] separately, and then

put that into a platform' (A3:65). Company C use product lines to leverage on invested test effort in many products. When changing the platform version 'we need to do the impact analysis for how things will be affected. And then we do the regression test on a basic functionality to see that no new faults have been introduced' (C3:55).

Process Enforcement Practices (P5)

External requirements and regulations on certain practices affect the motivation and incentive for enforcing processes and practices that support alignment. This is especially clear in Company C, which develops safety critical systems. 'Since it is safety-critical systems, we have to show that we have covered all the requirements, that we have tested them' (C1:6). It is admitted that traceability is costly, but, non-negotiable in their case. 'There is a lot of administration in that, in creating this matrix, but it has to be done. Since it is safety-critical systems, it is a reason for all the documentation involved' (C1:6). They also have an external assessor to validate that the processes are in place and are adhered to. An alternative enforcement practice was proposed by one interviewee from Company F (which does not operate in a safety-critical domain) who suggested that alignment could be achieved by enforcing traceability through integrating process enforcement in the development tools (F14:161) though this had not been applied.

Tracing Between Artefacts

The tracing practices between requirements and test artefacts vary over a large range of options from simple mappings between documents to extensive traces between detailed requirements and test cases.

- **Document-level traces (P6.1)** where links are retained between related documents is the simplest tracing practice. This is applied at Company A: 'we have some mapping there, between the project test plan and the project requirement specification. But this is a fragile link' (A2:69).
- **Requirement-test case traces (P6.2)** is the most commonly mentioned tracing practice where individual test cases are traced to individual requirements. This practice influences how test cases are specified: 'It is about keeping the test case a bit less complex and that tends to lead to keep them to single requirements rather than to several requirements' (F6:123).
- **Using test cases as requirements (P6.3)** where detailed requirements are documented as test cases is another option where the tracing become implicit at the detailed level when requirements and test cases are represented by the same entity. This practice was being introduced at Company F. 'At a certain level you write requirements, but then if you go into even more detail, what you are writing is probably very equivalent to a test case' (F5:113).

While this resolves the need for creating and maintaining traces at that level, these test-case requirements need to be aligned to requirements and testing information at higher abstraction levels. 'There will be teams responsible for mapping these test cases with the high-level requirements' (F10:150). Company A has this practice in place, though not pre-planned but due to test cases being better maintained over time than requirements. 'They know that this test case was created for this requirement some time ago [...and] implicitly [...] the database of test cases becomes a requirements specification' (A2:51).

- **Same abstraction levels used for requirements and test specifications (P6.4)** is an alignment practice related to the structure of information. First, the requirements information is structured according to suitable categories. The requirements are then detailed and documented within each category, and the same categorisation used for the test specifications. Company C has 'different levels of requirements specifications and test specifications, top level, sub-system, module level, and down to code' (C3:67), and Company D presents similar on the test processes and artefacts (D3:53). It is worth noting that both Company C and D develop safety-critical systems. At Company F, a project leader described 'the correlation between the different test [levels]' and different requirement levels; at the most detailed level 'test cases that specify how the code should work' and at the next level 'scenario test cases' (F8:16).

Practice of Traceability Responsible Role (P7)

For large projects, and for safety-critical projects, the task of creating and maintaining the traces may be assigned to certain roles. In Company E, one of the interviewees is responsible for consolidating the information from several projects to the main product level. 'This is what I do, but since the product is so big, the actual checking in the system is done by the technical coordinator for every project' (E3:54). In one of the companies with safety-critical projects this role also exists; 'a safety engineer [...] worked with the verification matrix and put in all the information [...] from the sub products tests in the tool and also we can have the verification matrix on our level' (C2:104).

Tool Support

Tool support is a popular topic on which everyone has an opinion when discussing alignment. The tool practices used for requirements and test management vary between companies, as does the tool support for tracing between these artefacts.

- **Tool support for requirements and test management (P8.1)** varies hugely among the companies in this study, as summarised in Table 5. Company A

Table 5: Tool usage for requirements and test cases, and for tracing between them. For Company F the tool set-up prior to the major process change are also given (marked with 'Fp').

	Reqts. Tool	**Tracing tool**	**Testing tool**
Requirements	C, D, E, Fp		F
Traces	C	D, E, Fp	F
Test cases	C		A, D, E, F, Fp

uses a test management tool, while requirements are stored as text. Companies D and E use a requirements management tool for requirements and a test management tool for testing. This was the previous practice at Company F too. Company C uses a requirements management tool for both requirements and test, while Company F aims to start using a test management tool for both requirements and testing. Most of the companies use commercial tools, though Company A has an in-house tool, which they describe as 'a version handling system for test cases' (A2:208).

- **Tool support for requirements-test case tracing (P8.2)** is vital for supporting traceability between the requirements and test cases stored in the tools used for requirements and test management. Depending on the tool usage, tracing needs to be supported either within a tool, or two tools need to be integrated to allow tracing between them. For some companies, *only manual tracing is supported*. For example, at Company D a systems architect describes that it is possible to 'trace requirements between different tools such as [requirements] modules and Word documents' (D3:45). However, a software manager at the same company mentions problems in connecting the different tools and says 'the tools are not connected. It's a manual step, so that's not good, but it works' (D1:111).

  Tracing within tools is practiced at Company C where requirements and test cases are both stored in a commercial requirements management tool: 'when we have created all the test cases for a certain release, then we can automatically make this matrix show the links between [system] requirements and test cases' (C1:8). Company F has used the *between-tools practice* 'The requirements are synchronised over to where the test cases are stored' (F5:19). However, there are issues related to this practice. Many-to-many relationships are difficult to handle with the existing tool support (F2:167). Furthermore, relationships at the same level of detail are easier to handle than across different abstraction levels. One requirements engineer asks for 'a tool that connects everything; your requirement with design documents with test cases with your code maybe even your planning document' (F5:17). In a large, complex system and its development organisation, there

is a need for 'mapping towards all kinds of directions-per function group, per test cases, and from the requirement level' (F11:139).

Many interviewees had complaints about their tools, and the integration between them. Merely having tool support in place is not sufficient, but it must be efficient and usable. For example, Company E have tools for supporting traceability between requirements and test state of connected test cases but 'we don't do it because the tool we have is simply not efficient enough' (E3:57) to handle the test state for the huge amount of verified variants. Similarly, at Company E the integration solution (involving a special module for integrating different tools) is no longer in use and they have reverted to manual tracing practices: 'In some way we are doing it, but I think we are doing it manually in Excel sheets' (E2:49).

Finally, companies moving from waterfall processes towards agile practices tend to find their tool suite too heavy weight for the new situation (E3:89). Users of these tools not only include engineers, but also management, which implies different demands. A test manager states: 'Things are easy to do if you have a lot of hands on experience with the tools but what you really need is something that the [higher level] managers can use' (F10:249).

Alignment Metrics (P9)

Measurements can be used to gain control of the alignment between requirements and testing. The most commonly mentioned metrics concern test case coverage of requirements. For example, Company C 'measure[s] how many requirements are already covered with test cases and how many are not' (C1:64). These metrics are derived from the combined requirements and test management tool. Companies E and F have a similar approach, although with two different tools. They both point out that, in addition to the metrics, it is a matter of judgement to assess full requirements coverage. 'If you have one requirement, that requirement may need 16 test cases to be fully compliant. But you implement only 14 out of those. And we don't have any system to see that these 2 are missing' (E3:81). And, 'just because there are 10 test cases, we don't know if [the requirement] is fully covered' (F11:34).

Furthermore, there is a versioning issue to be taken into account when assessing the requirements coverage for verification. 'It is hard to say if it [coverage] should be on the latest software [version] before delivery or...?' (F10:224) The reverse relationship of requirements coverage of all test cases is not always in place or measured. 'Sometimes we have test cases testing functionality not specified in the requirements database' (F11:133). Other alignment metrics were mentioned, for example, missing links between requirements and tests, number of requirements at different levels (F5:112), test costs for changed requirements (F14:205), and requirements review status (F14:205). Not all of these practices were prac-

ticed at the studied companies even though some mentioned that such measures would be useful (F14:219).

Job Rotation Practices (P10)

Job rotation was suggested in interviews at Companies D and F as a way to improve alignment by extending contact networks and experiences across departments and roles, and thereby supporting spreading and sharing perspectives within an organisation. In general, the interviews revealed that alignment is very dependent on individuals, their experience, competence and their ability to communicate and align with others. The practice of job rotation was mentioned as a proposal for the future and not currently implemented at any of the included companies.

4.3 Practices that Address the Challenges

This section provides an overview of the relationships between the alignment challenges and practices identified in this study (and reported in Sections 4.1 and 4.2). The mapping is intended as an initial guide for practitioners in identifying practices to consider in addressing the most pressing alignment challenges in their organisations. The connections have been derived through analysis of the parts of the interview transcripts connected to each challenge and practice, summarised in Figure 6 and elaborated next. The mapping clearly shows that there are many-to-many relations between challenges and practices. There is no single practice that solves each challenge. Consequently, the mapping is aimed at a strategic level of improvement processes within a company, rather than a lower level of practical implementation. After having assessed the challenges and practices of REVV alignment within a company, the provided mapping can support strategic decisions concerning which areas to improve. Thereafter, relevant practices can be tailored for use within the specific context. Below we discuss our findings, challenge by challenge.

The practices observed to address the challenge of having **common goals within an organisation (Ch1)** mainly concern increasing the synchronisation and communication between different units and roles. This can be achieved through involving customers and development-near roles in the requirements process (P1.1, P1.2, P1.3, P1.5); documenting requirement decision rationale (P1.6); validating requirements through test case reviews (P2.1) and product managers reviewing prototypes (P2.3); and involving testing roles in change management (P4.1). Goal alignment is also increased by the practice of basing launch decisions made by management on test reports (P2.4) produced by testers. Furthermore, tracing between artefacts (P6.1-6.4) provides a technical basis for supporting efficient communication of requirements. Job rotation (P10) is mentioned as a long-term practice for sharing goals and synchronising perspectives across the organisation. In the mid-term perspective, customer feedback received by testers for previous

Table 6: Mapping of practices to the challenges they are found to address. An S represents a suggested, but not implemented practice.

	P1 RE practices	P2 Validation practices	P3 Verification practices	P4 Change management	P5 Process enforcement	P6 Tracing between artefacts	P7 Traceability responsibility role	P8 Tool practices	P9 Alignment metrics	P10 Job rotation
Ch1 Aligning goals and perspectives within organisation	P1.1–1.3, 1.5–1.6	P2.1, 2.3–2.4	P3.3	P4.1		P6.1–6.4				P10 (S)
Ch2 Cooperating successfully	P1.2–1.3, 1.5–1.6	P2.1, 2.3, 2.4	P3.1	P4.1						P10 (S)
Ch3 Requirements specification quality	P1.1–1.5	P2.1, 2.5		P4.1	P5	P6.2–6.3			P9	
Ch4 VV quality	P1.1–1.5	P2.1–2.3, 2.5	P3.1–3.3		P5	P6.1–6.4			P9	
Ch5 Maintaining alignment when requirements change		P2.2, P2.5		P4.1–4.2	P5	P6.1–6.4	P7		P9	
Ch6 Requirements abstraction levels	P1.1, 1.6					P6.4				
Ch7 Traceability		P2.1			P5	P6.1–6.4	P7	P8.1–8.2	P9	
Ch8 Time and resource availability				P4.1	P5					
Ch9 Managing a large document space						P6.1–6.4	P7	P8.1–8.2	P9	
Ch10 Outsourcing of components or testing	P1.1–1.5	P2.1–2.3	P3.4			P6.4				

A blank cell indicates that no connection was mentioned during the interviews

projects (P3.3) can be reused as input when defining roadmaps and products plans thereby further coordinating the testers with the requirements engineers responsible for the future requirements.

The challenge of **cooperating successfully (Ch2)** is closely related to the first challenge (Ch1) as being a means to foster common goals. Practices to achieve close cooperation across roles and organisational borders hence include cross-functional involvement (P1.2, P1.5, P2.4) and reviews (P1.3, P2.1, P2.3), feedback through early and continuous test activities (P3.1), as well as, joint decisions about changes in change control boards (P4.1) and documenting requirement decision rationales (P1.6). The former are practices are embraced in agile processes, while the latter practices of change control boards and documentation of rationales were removed for the studied cases when agile processes were introduced. Job rotation (P10), with its general contribution to building networks, is expected to facilitate closer cooperation across organisational units and between roles.

The challenge of achieving good **requirements specification quality (Ch3)** is primarily addressed by the practices for requirements engineering (P1.1-1.5), validation (P2.1, P2.5) and managing requirement changes (P4.1). Some of the traceability practices (P6.2, P6.3) also address the quality of requirements in terms of being well structured and defined at the right level of detail. Furthermore, awareness of the importance of alignment and full requirements coverage may induce and enable organisations in producing better requirements specifications. This awareness can be encouraged with the use of alignment metrics (P9) or enforced (P5) through regulations for safety-critical software and/or by integrating process adherence in development tools.

The challenge of achieving good **validation and verification quality (Ch4)** is addressed by practices to ensure clear and agreed requirements, such as cross-functional reviews (P1.3, P2.1), involving development roles in detailing requirements (P1.2) and customers in defining acceptance criteria (P2.2). Validation is supported by product managers reviewing prototypes (P2.3) and by user/customer testing (P2.5). Verification is improved by early verification activities (P3.1) and through independent testing (P3.2) where testers are not influenced by other engineers' interpretation of the requirements. Complete and up-to-date requirements information is a prerequisite for full test coverage, which can be addressed by requirements engineering practices (P1.1-1.5), testers re-using customer feedback (P3.3) (rather than incorrect requirements specification) and indirectly by traceability practices (P6.1-6.4). The external enforcement (P5) of the full test coverage and alignment metrics (P9) are practices that provide incentives for full test coverage including quality requirements.

Maintaining alignment when requirements change (Ch5) is a challenge that clearly connects to change and traceability practices (P4.1-4.2, P6.1-6.4 and P7). However, also the validation practices of having acceptance tests based on user scenarios (P2.2) and user/customer testing (P2.5) address this challenge by providing feedback on incorrectly updated requirements, test cases and/or software.

Furthermore, having alignment metrics in place (P9) and external regulations on documentation and traceability (P5) is an incentive to maintain alignment as requirements change.

The challenge of managing **requirements abstraction levels (Ch6)** is addressed by the requirements practice of including the customer in requirements work throughout a project (P1.1) and the tracing practices of matching abstractions levels for requirements and test artefacts (P6.4). Both of these practices exercise the different requirements levels and thereby support identifying mismatches. This challenge is also supported by documentation of requirement decision rationales (P1.6) by providing additional requirements information to the roles at the different abstraction level.

Traceability (Ch7) in itself is identified as a challenge in the study, and interviewees identified practices on the information items to be traced (P6.1-6.4), as well as, tools (P8.1-8.2) to enable tracing. In addition, the practice of reviewing test cases against requirements (P2.1) may also support identifying sufficient and/or missing traces. Furthermore, requirements coverage metrics (P9) are proposed as a means to monitor and support traceability. However, as noticed by Companies E and F, simple coverage metrics are not sufficient to ensure ample alignment. Process enforcement practices (P5) and assigning specific roles responsible for traceability (P7) are identified as key practices in creating and maintaining traces between artefacts.

The practical aspects of the challenge on **availability of time and resources (Ch8)** are mainly a matter of project management practices, and hence not directly linked to the alignment practices. However, the practice of involving testing roles in the change management process (P4.1) may partly mitigate this challenge by supporting an increased awareness of the verification cost and impact of changes. Furthermore, in companies for which alignment practices are externally enforced (P5) there is an awareness of the importance of alignment of software development, but also an increased willingness to take the cost of alignment including tracing.

The **large document space (Ch9)** is a challenge that can be partly addressed with good tool support (P8.1-8.2) and tracing (P6.1-6.4, P7) practices. The study specifically identifies that a tool that fits a medium-sized project may be very hard to use in a large one. One way of getting a synthesised view of the level of alignment between large sets of information is to characterise it, using quantitative alignment measurements (P9). It does not solve the large-scale problem, but may help assess the current status and direct management attention to problem areas.

Outsourcing (Ch10) is a challenge that is related to timing, which is a project management issue, and to communication of the requirements, which are to be developed or tested by an external team. The primary practice to apply to outsourcing is customer communication (P1.1). Frequent and good communication can ensure a common perspective and direction, in particular in the early project phases. In

addition, other practices for improved cooperation (P1.2-1.5, P2.1-2.3) are even more important when working in different organisational units, times zones, and cultural contexts. Furthermore, in an outsourcing situation the requirements specification is a key channel of communication, often also in contractual form. Thus, having requirements and tests specified at the same level of abstraction (P6.4), feasible for the purpose, is a practice to facilitate the outsourcing. Finally, training the outsourced or off-shored team (P3.4) in company practices and tools also addresses this challenge.

In summary, the interviewees brought forward practices, which address some of the identified challenges in aligning requirements and testing. The practices are no quick-fix solutions, but the mapping should be seen as a guideline to recommend areas for long-term improvement, based on empirical observations of industry practice.

5 Discussion

Alignment between requirements and test ranges not only the life-cycle of a software development project, but also company goals and strategy, and affects a variety of issues, from human communication to tools and their usage. Practices differ largely between companies of varying size and maturity, domain and product type, etc. One-size alignment practices clearly do not fit all.

A wide collection of alignment challenges and practices have been identified based on the large amount of experiences represented by our 30 interviewees from six different companies, covering multiple roles, domains and situations. Through analysing this data and deriving results from it, the following general observations have been made by the researchers:

1. The human and organisational sides of software development are at the core of industrial alignment practices.
2. The requirements are the frame of reference for the software to be built, and hence the quality of the requirements is critical for alignment with testing activities.
3. The large difference in size (factor 20) between the companies, in combination with variations in domain and product type, affects the characteristics of the alignment challenges and applicable practices.
4. The incentives for investing in good alignment practices vary between domains.

Organisational and human issues are related to several of the identified challenges (Ch1, Ch2, Ch8, and Ch10). Successful cooperation and collaboration (Ch2) is a human issue. Having common goals and perspectives for a development project is initially a matter of clear communication of company strategies

and goals, and ultimately dependant on human-to-human communication (Ch1). Failures to meet customer requirements and expectations are often related to misunderstanding and misconception; a human failure although technical limitations, tools, equipment, specifications and so on, also play a role. It does not mean that the human factor should be blamed in every case and for each failure. However, this factor should be taken into account when shaping the work conditions for software engineers. These issues become even more pressing when outsourcing testing. Jones *et al.* [252] found that failure to align outsourced testing activities with in-house development resulted in wasted effort, mainly due to weak communication of requirements and changes of them.

Several of the identified alignment practices involve the human and organisational side of software engineering. Examples include communication practices with customers, cross-role and cross-functional meetings in requirements elicitation and reviews, communication of changes, as well as, a proposed job rotation practice to improve human-to-human communication. This confirms previous research that alignment can be improved by increasing the interaction between testers and requirements engineers. For example, including testers early on and, in particular, when defining the requirements, can lead to improved requirements quality [458]. However, Uusitalo *et al.* also found that cross collaboration can be hard to realise due to unavailability of requirements owners and testers on account of other assignments and distributed development [458]. In general, processes and roles that support and enforce the necessary communication paths may enhance alignment. For example, Paci *et al.* [365] report on a process for handling requirements changes through clearly defined communication interfaces. This process relies on roles propagating change information within their area, rather than relying on more general communication and competence [365]. This also confirms the findings of Uusitalo *et al.* that increased cross communication reduces the amount of assumptions made by testers on requirements interpretation, and results in an increased reliability of test results and subsequent products [458]. Similarly, Fogelström and Gorschek [184] found that involving testers as reviewers through test-case driven inspections of requirements increases the interaction with requirements-related roles and can improve the overall quality of the requirements, thereby supporting alignment. Furthermore, even technical practices, such as tool support for requirements and test management, clearly have a human side concerning degree of usability and usefulness for different groups of stakeholders in an organisation.

Defining requirements of good quality (Ch3) is central to enabling good alignment and coordination with other development activities, including validation and verification. This challenge is not primarily related to the style of requirements, whether scenario based, plain textual, or formal. But, rather the quality characteristics of the requirements are important, i.e. being verifiable, clear, complete, at a suitable level of abstraction and up-to-date. This relates to results from an empirical study by Ferguson and Lami [181] that found that unclear require-

ments have a higher risk of resulting in test failures. A similar reverse relationship is reported by Graham [205], that clearer and verifiable requirements enable testers to define test cases that match the intended requirements. In addition, Uusitalo *et al.* [458] found that poor quality of requirements was a hindrance to maintaining traces from test cases. Sikora *et al.* [431] found that requirements reviews is the dominant practice applied to address quality assurance of the requirements for embedded systems and that industry need additional and improved techniques for achieving good requirements quality. Furthermore, requirements quality is related to involving, not only requirements engineers in the requirements engineering, but also VV roles in early stages. This can be achieved by involving non-RE roles in reviews and in detailing requirements. This also contributes to cross-organisational communication and learning, and supports producing requirements that are both useful and used. Uusitalo *et al.* [458] found that testers have a different viewpoint that makes them well suited to identifying deficiencies in the requirements including unverifiability and omissions. Martin and Melnik [330] take this approach one step further by suggesting that the requirements themselves be specified as acceptance test cases, which are then used to verify the behaviour of the software. This approach was evaluated through an experiment by Ricca *et al.* [396] who found that this helped to clarify and increase the joint understanding of requirements with substantially the same amount of effort. Furthermore, our findings that RE practices play a vital role in supporting REVV alignment confirm previous conclusions that the requirements process is an important enabler for testing activities and that RE improvements can support alignment with testing [458].

Company size varies largely between the six companies in this study. Similarly, the challenges and practices also vary between the companies. While smaller project groups of 5-10 persons can handle alignment through a combination of informal and formal project meetings. Large-scale projects require more powerful process and tool support to ensure coordination and navigation of the (larger) information space between different phases and hierarchies in the organisation. This was illustrated by different views on the same state-of-the-art requirements management tool. The tool supported alignment well in one medium-sized project (Company C), but was frequently mentioned by the interviewees for the largest company (Company F) as a huge alignment challenge.

In some cases (e.g. Company F), agile practices are introduced to manage large projects by creating several smaller, self-governing and less dependent units. Our study shows that this supports control and alignment at the local level, but, at the expense of global control and alignment (Company E). The size-related alignment challenges then re-appear in a new shape, at another level in the organisation. For example, granting development teams mandate to define and change detailed requirements increases speed and efficiency at the team level, but increases the challenge of communicating and coordinating these changes wider within a large organisation.

The incentives for applying alignment practices, specifically tracing be-

tween requirements and test artefacts, vary across the studied companies. Applying alignment practices seems to be connected to the incentives for enforcing certain practices, such as tracing and extensive documentation. The companies reporting the most rigid and continuously maintained alignment practices are those working in domains where customers or regulatory bodies require such practices. Both of these companies (C and D) have enforced alignment practices in their development including tracing between requirements and tests. Interestingly these are also the companies in our study which apply a traditional and rigorous development model. It is our interpretation that the companies with the most agile, and least rigorous, development processes (A and B) are also the companies which rely heavily on people-based alignment and tracing, rather than on investing in more structured practices. These are also the two companies that do not have tracing between artefacts in place, even partially. While for the remaining companies (E and F) which apply agile-inspired processes, but with structured elements (e.g. eRUP), traceability is in place partly or locally. Our interpretation of the relationship between the included companies concerning incentives and degree of rigour in applying structured alignment practices is illustrated in Figure 4 together with the relative size of their software development. The observed connection between degree of rigour and incentives for alignment are similar to other findings concerning safety-critical development. Namely, that alignment is enabled by more rigorous practices such as concurrently designed processes [286] or model-based testing [214, 356]. Furthermore, companies in safety-critical domains have been found to apply more rigorous processes and testing practices [411]. In contrast, neglect of quality requirements, including safety aspects has been found to one of the challenges of agile RE [94].

Interestingly, several alignment challenges (e.g. tracing, communicating requirements changes) were experienced also for the companies developing safety-critical software (C and D) despite having tracing in place and applying practices to mitigate alignment challenges (e.g. frequent customer communication, tracing responsible role, change management process involving testers etc.) This might be explained by a greater awareness of the issues at hand, but also that the increased demands posed by the higher levels of quality demands requires additional alignment practice beyond those needed for non-critical software.

When the documentation and tracing practices are directly enforced from outside the organisation, they cannot be negotiated and the cost has to be taken [468]. In organisations without these external requirements the business case for investing in these practices needs to be defined, which does not seem to be the case for the studied organisations. Despite the existence of frustration and rework due to bad alignment, the corresponding costs are seldom quantified at any level. Improving alignment involves short term investments in tools, work to recapture traces between large legacies of artefacts, and/or in changed working practices. The returns on these investments are gained mainly in the longer term. This makes it hard to put focus and priority on alignment practices in a short-sighted financial and man-

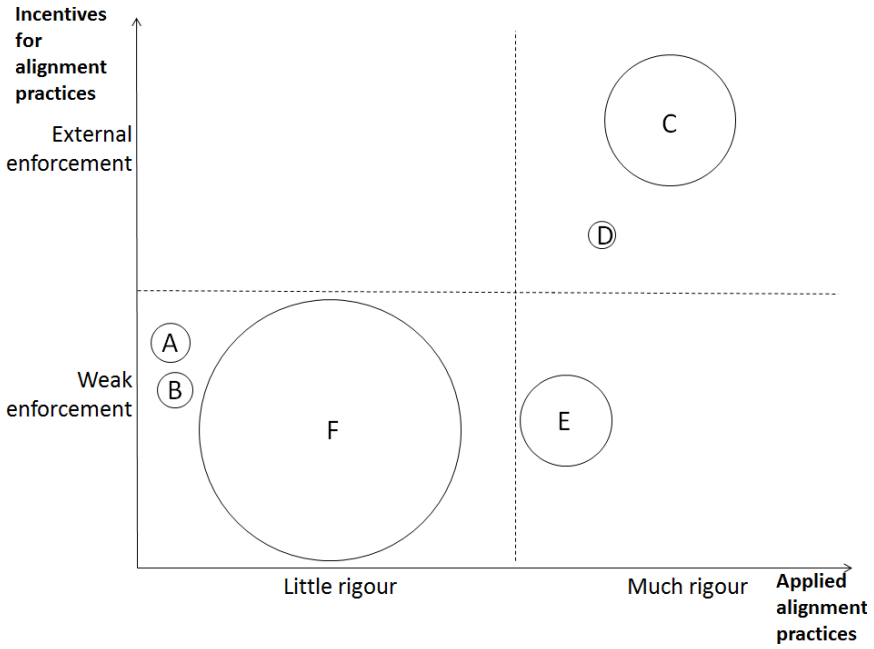


Figure 4: Rough overview of the relationship between the variation factors size, rigour in applying alignment practices and incentive for alignment practices for the studied companies. Number of people in software development is reflected by the relative size of the circle.

agement culture. Finally, requirements volatility increases the importance and cost to achieve REVV alignment. This need to manage a rate of requirements changes often drives the introduction of agile practices. These practices are strong in team cooperation, but weak in documentation and traceability between artefacts. The companies (C and D) with lower requirements volatility and where development is mainly plan-driven and bespoken, have the most elaborated documentation and traceability practices. In both cases, the practices are enforced by regulatory requirements. However, in our study, it is not possible to distinguish between the effects of different rates of change and the effects of operating in a safety-critical domain with regulations on documentation and traceability.

In summary, challenges and practices for REVV alignment span the whole development life cycle. Alignment involves the human and organisational side of software engineering and requires the requirements to be of good quality. In addition, the incentives for alignment greatly vary between companies of different size and application domain. Future research and practice should consider these variations in identifying suitable practices for REVV alignment, tailored to different domains and organisations.

6 Conclusions

Successful and efficient software development, in particular on the large scale, requires coordination of the people, activities and artefacts involved [133, 134, 283]. This includes alignment of the areas of requirements and test [133, 286, 414, 458]. Methods and techniques for linking artefacts abound including tracing and use of model-based engineering. However, companies experience challenges in achieving alignment including full traceability. These challenges are faced also by companies with strong incentives for investing in REVV alignment such as for safety critical software where documentation and tracing is regulated. This indicates that the underlying issues lie elsewhere and require aligning of not only the artefacts, but also of other factors. In order to gain a deeper understanding of the industrial challenges and practices for aligning RE with VV, we launched a case study covering six companies of varying size, domain, and history. This paper reports the outcome of that study and provides a description of the industrial state of practice in six companies. We provide categorised lists of (RQ1) industrial alignment challenges and (RQ2) industrial practices for improving alignment, and (RQ3) a mapping between challenges and practices. Our results, based on 30 interviews with different roles in the six companies, add extensive empirical input to the existing scarce knowledge of industrial practice in this field [414, 458]. In addition, this paper presents new insights into factors that explain needs and define solutions for overcoming REVV alignment challenges.

We conclude with four high-level observations on the alignment between requirements and testing. Firstly, as in many other branches of software engineering,

the *human side* is central, and communication and coordination between people is vital, so also between requirements engineers and testers, as one interviewee said: 'start talking to each other!' (F7:88) Further, the *quality and accuracy of the requirements* is a crucial starting point for testing the produced software in-line with the defined and agreed requirements. Additionally, the *size of the development organisation* and its projects is a key variation factor for both challenges and practices of alignment. Tools and practices may not be scalable, but rather need to be selected and tailored to suit the specific company, size and domain. Finally, alignment practices such as good requirements documentation and tracing seem to be applied for safety-critical development through external enforcement. In contrast, for non-safety critical cases only internal *motivation* exists for the alignment practices even though these companies report facing large challenges caused by misalignment such as incorrectly implemented requirements, delays and wasted effort. For these cases, support for assessing the cost and benefits of REVV alignment could provide a means for organisations to increase the awareness of the importance of alignment and also tailor their processes to a certain level of alignment, suitable and cost effective for their specific situation and domain.

In summary, our study reports on the current practice in several industrial domains. Practical means are provided for recognising challenges and problems in this field and matching them with potential improvement practices. Furthermore, the identified challenges pose a wide assortment of issues for researchers to address in order to improve REVV alignment practice, and ultimately the software engineering practices.

Acknowledgement

We want to thank Børje Haugset for acting as interviewer in three of the interviews. We would also like to thank all the participating companies and anonymous interviewees for their contribution to this project. The research was funded by EASE Industrial Excellence Center for Embedded Applications Software Engineering (http://ease.cs.lth.se).

CHAPTER II

RECOVERING FROM A DECADE: A SYSTEMATIC REVIEW OF INFORMATION RETRIEVAL APPROACHES TO SOFTWARE TRACEABILITY

Abstract

Context: Engineers in large-scale software development have to manage large amounts of information, spread across many artifacts. Several researchers have proposed expressing retrieval of trace links among artifacts, i.e. trace recovery, as an Information Retrieval (IR) problem. **Objective:** The objective of this study is to produce a map of work on IR-based trace recovery, with a particular focus on previous evaluations and strength of evidence. **Method:** We conducted a systematic mapping of IR-based trace recovery. **Results:** Of the 79 publications classified, a majority applied algebraic IR models. While a set of studies on students indicate that IR-based trace recovery tools support certain work tasks, most previous studies do not go beyond reporting precision and recall of candidate trace links from evaluations using datasets containing less than 500 artifacts. **Conclusions:** Our review identified a need of industrial case studies. Furthermore, we conclude that the overall quality of reporting should be improved regarding both context and tool details, measures reported, and use of IR terminology. Finally, based on our empirical findings, we present suggestions on how to advance research on IR-based trace recovery.

Markus Borg, Per Runeson, and Anders Ardö, *Empirical Software Engineering*, 19(6), pp. 1565-1616, 2014

1 Introduction

The successful evolution of software systems involves concise and quick access to information. However, information overload plagues software engineers, as large amounts of formal and informal information is continuously produced and modified [111, 362]. Inevitably, especially in large-scale projects, this leads to a challenging information landscape, that includes, apart from the source code itself, requirements specifications of various abstraction levels, test case descriptions, defect reports, manuals, and the like. The state-of-practice approach to structure such information is to organize artifacts in databases, e.g. document management systems, requirements databases, and code repositories, and to manually maintain trace links [204, 231]. With access to trace information, engineers can more efficiently perform work tasks such as impact analysis, identification of reusable artifacts, and requirements validation [18, 473]. Furthermore, research has identified lack of traceability to be a major contributing factor in project overruns and failures [111, 160, 204]. Moreover, as traceability plays a role in software verification, safety standards such as ISO 26262 [242] for the automotive industry, and IEC 61511 [238] for the process industry, mandate maintenance of traceability information [264], as does the CMMI process improvement model [97]. However, manually maintaining trace links is an approach that does not scale [216]. In addition, the dynamics of software development makes it tedious and error-prone [160, 175, 231].

As a consequence, engineers would benefit from additional means of dealing with information seeking and retrieval, to navigate effectively the heterogeneous information landscape of software development projects. Several researchers have claimed it feasible to treat traceability as an information retrieval (IR) problem [18, 141,231,315,326]. Also, other studies have reported that the use of semi-automated trace recovery reduces human effort when performing requirements tracing [139, 144, 148, 231, 355]. The IR approach builds upon the assumption that if engineers refer to the same aspects of the system, similar language is used across different software artifacts. Thus, tools suggest trace links based on Natural Language (NL) content. During the first decade of the millennium, substantial research effort has been spent on tailoring, applying, and evaluating IR techniques to software engineering, but we found that a comprehensive overview of the field is missing. Such a secondary analysis would provide an evidence based foundation for future research, and advise industry practice [273]. As such, the gathered empirical evidence could be used to validate, and possibly intensify, the recent calls for future research by the traceability research community [202], organized by the Center of Excellence for Software Traceability (CoEST)[1]. Furthermore, it could assess the recent claims that applying more advanced IR models does not improve results [175, 360].

[1] www.coest.org

We have conducted a Systematic Mapping (SM) study [272, 373] that clusters publications on IR-based trace recovery. SMs and Systematic Literature Reviews (SLR) are primarily distinguished by their driving Research Questions (RQ) [272], i.e. an SM identifies research gaps and clusters evidence to direct future research, while an SLR synthesizes empirical evidence on a specific RQ. The rigor of the methodologies is a key asset in ensuring a comprehensive collection of published evidence. We define our overall goals of this SM in three RQs:

RQ1 Which IR models and enhancement strategies have been most frequently applied to perform trace recovery among NL software artifacts?

RQ2 Which types of NL software artifacts have been most frequently linked in IR-based trace recovery studies?

RQ3 How strong is the evidence, wrt. degree of realism in the evaluations, of IR-based trace recovery?

This paper is organized as follows. Section 2 contains a thorough definition of the IR terminology we refer to throughout this paper, and a description of how IR tools can be used in a trace recovery process. Section 3 presents related work, i.e. the history of IR-based trace recovery, and related secondary and methodological studies. Section 4 describes how the SM was conducted. Section 5 shows the results from the study. Section 6 discusses our research questions based on the results. Finally, Section 7 presents a summary of our contributions and suggests directions for future research.

2 Background

This section presents fundamentals of IR, and how tools implementing IR models can be used in a trace recovery process.

2.1 IR Background and Terminology

As the study identified variations in use of terminology, this section defines the terminology used in this study (summarized in Table 1), which is aligned with recently redefined terms [113]. We use the following IR definition: "*information retrieval is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers)*" [325]. If a retrieved document satisfies such a need, we consider it relevant. We solely consider text retrieval in the study, yet we follow convention and refer to it as IR. In our interpretation, the starting point is that any approach that retrieves documents relevant to a query qualifies as IR. The terms Natural Language Processing (NLP) and Linguistic Engineering (LE) are used in a subset of the mapped publications of this study, even if they refer to the same

IR techniques. We consider NLP and LE to be equivalent and borrow two definitions from Liddy [306]: "*NL text is text written in a language used by humans to communicate to one another*", and "*NLP is a range of computational techniques for analyzing and representing NL text*". As a result, IR (referring to a process solving a problem) and NLP (referring to a set of techniques) are overlapping. In contrast to the decision by Falessi *et al.* [175] to consistently apply the term NLP, we choose to use IR in this study, as we prefer to focus on the process rather than the techniques. While trace recovery truly deals with solutions targeting NL text, we prefer to primarily consider it as a problem of satisfying an information need.

Furthermore, a "*software artifact is any piece of information, a final or intermediate work product, which is produced and maintained during software development*" [284], e.g. requirements, design documents, source code, test specifications, manuals, and defect reports. To improve readability, we refer to such pieces of information only as 'artifacts'. Regarding traceability, we use two recent definitions: "*traceability is the potential for traces to be established and used*" and "*trace recovery is an approach to create trace links after the artifacts that they associate have been generated or manipulated*" [113]. In the literature, the trace recovery process is referred to in heterogeneous ways including traceability link recovery, inter-document correlation, document dependency/similarity detection, and document consolidation. We refer to all such approaches as *trace recovery*, and also use the term *links* without differentiating between dependencies, relations and similarities between artifacts.

In line with previous research, we use the term *dataset* to refer to the set of artifacts that is used as input in evaluations and *preprocessing* to refer to all processing of NL text before the IR models (discussed next) are applied [36], e.g. stop word removal, stemming and identifier (ID) splitting names expressed in CamelCase (i.e. identifiers named according to the coding convention to capitalize the first character in every word) or identifiers named according to the under_score convention. *Feature selection* is the process of selecting a subset of terms to represent a document, in an attempt to decrease the size of the effective vocabulary and to remove noise [325].

To support trace recovery, several IR models have been applied. Since we identified contradicting interpretations of what is considered a model, weighting scheme, and similarity measure, we briefly present our understanding of the IR field. IR models often apply the *Bag-of-Words* (BoW) model, a simplifying assumption that represents a document as an unordered collection of words, disregarding word order [325]. Most existing IR models can be classified as either algebraic or probabilistic, depending on how relevance between queries and documents is measured. In algebraic IR models, relevance is assumed to be correlated with similarity [490]. The most well-known algebraic model is the commonly applied *Vector Space Model (VSM)* [417], which due to its many variation points acts as a framework for retrieval. Common to all variations of VSM is that both documents and queries are represented as vectors in a high-dimensional space (every

term, after preprocessing, in the document collection constitutes a dimension) and that similarities are calculated between vectors using some distance function. Individual terms are not equally meaningful in characterizing documents, thus they are weighted accordingly. Term weights can be both binary (i.e. existing or non-existing) and raw (i.e. based on term frequency) but usually some variant of *Term Frequency-Inverse Document Frequency* (TF-IDF) weighting is applied. TF-IDF is used to weight a term based on the length of the document and the frequency of the term, both in the document and in the entire document collection [433]. Regarding *similarity measures*, the cosine similarity (calculated as the cosine of the angle between vectors) is dominating in IR-based trace recovery using algebraic models, but also Dice's coefficient and the Jaccard index [325] have been applied. In an attempt to reduce the noise of NL (such as synonymy and polysemy), *Latent Semantic Indexing* (LSI) was introduced [150]. LSI reduces the dimensions of the vector space, finding semi-dimensions using singular value decomposition. The new dimensions are no longer individual terms, but concepts represented as combinations of terms. In the VSM, *relevance feedback* (i.e. improving the query based on human judgement of partial search results, followed by re-executing an improved search query) is typically achieved by updating the query vector [490]. In IR-based trace recovery, this is commonly implemented using the Standard Rocchio method [408]. The method adjusts the query vector toward the centroid vector of the relevant documents, and away from the centroid vector of the non-relevant documents.

In probabilistic retrieval, relevance between a query and a document is estimated by probabilistic models. The IR is expressed as a classification problem, documents being either relevant or non-relevant [433]. Documents are then ranked according to their probability of being relevant [329], referred to as the probabilistic ranking principle [397]. In trace recovery, the *Binary Independence Retrieval* model (BIM) [398] was first applied to establish links. BIM naïvely assumes that terms are independently distributed, and essentially applies the Naïve Bayes classifier for document ranking [300]. Different weighting schemes have been explored to improve results, and currently the *BM25* weighting used in the non-binary Okapi system [400] constitutes state-of-the-art.

Another category of probabilistic retrieval is based on the model of an inference process in a *Probabilistic Inference Network* (PIN) [455]. In an inference network, relevance is modeled by the uncertainty associated with inferring the query from the document [490]. Inference networks can embed most other IR models, which simplifies the combining of approaches. In its simplest implementation, a document instantiates every term with a certain strength and multiple terms accumulate to a numerical score for a document given each specific query. Relevance feedback is possible also for BIM and PIN retrieval [490], but we have not identified any such attempts within the trace recovery research.

In the last years, another subset of probabilistic IR models has been applied to trace recovery. *Statistical Language Models* (LM) estimate an LM for each

document, then documents are ranked based on the probability that the LM of a document would generate the terms of the query [378]. A refinement of simple LMs, topic models, describes documents as a mixture over topics. Each individual topic is then characterized by an LM [491]. In trace recovery research, studies applying the four topic models *Probabilistic Latent Semantic Indexing* (PLSI) [221], *Latent Dirichlet Allocation* (LDA) [65], *Correlated Topic Model* (CTM) [64] and *Relational Topic Model* (RTM) [101] have been conducted. To measure the distance between LMs, where documents and queries are represented as stochastic variables, several different measures of distributional similarity exist, such as the *Jensen-Shannon divergence* (JS). To the best of our knowledge, the only implementation of relevance feedback in LM-based trace recovery was based on the *Mixture Model method* [492].

Several attempts are made to improve an IR model, in this paper referred to as *enhancement strategies*. Apart from the already described relevance feedback, one of the most common approaches in IR is to introduce a *thesaurus*. A thesaurus is a tool for vocabulary control, typically defined for a specific subject area, such as art or biology, formally organized so that *a priori* relationships between concepts are made explicit [7]. Standard usage of a thesaurus is to provide an IR system with *preferred and non-preferred terms*, restricted vocabularies of terms that the IR system is allowed to, or not allowed to, use for indexing and searching, and *semantic relations*, relations between terms such as synonymy and hyponymy. Another enhancement strategy in IR is *phrasing*, an approach to exceed indexing according to the BoW model [122]. A phrase is a sequence of two or more words, expected to be more accurate in representing document content than independent words. Detecting phrases for indexing can be done using either a statistical analysis of term frequency and co-occurrence, or by a syntactical approach, i.e. analyzing grammatical structure using a parts-of-speech tagger. Yet another enhancement strategy is *clustering*, based on the hypothesis that "*documents relevant to a query tend to be more similar to each other than to irrelevant documents and hence are likely to be clustered together*" [102]. Clustering can be used for different purposes, e.g. presenting additional search results or to structure the presentation of search results.

Finally, a number of measures used to evaluate IR tools have been defined. Accuracy of a set of search results is primarily measured by the standard IR-measures *precision* (the fraction of retrieved instances that are relevant), *recall* (the fraction of relevant instances that are retrieved) and *F-measure* (harmonic mean of precision and recall, possibly weighted to favour one over another) [36]. Precision and recall values (P-R values) are typically reported pairwise or as precision and recall curves (P-R curves). Two other set-based measures, originating from the traceability community, are *Recovery Effort Index* (REI) [18] and *Selectivity* [443]. *Secondary measures* aim to go further than comparing sets of search results, and also consider their internal ranking. Two standard IR measures are *Mean Average Precision* (MAP) of precision scores for a query [325], and *Discounted Cumulative*

Table 1: A summary of fundamental IR terms applied in trace recovery. Note that only the vertical organization carries a meaning.

Retrieval Models			Misc.		
Algebraic models	**Probabilistic models**	**Statistical language models**	**Weighting schemes**	**Similarity measures / distance functions**	**Enhancement strategies**
Vector Space Model (VSM)	Binary Independence Model (BIM)	Language Model (LM)	Binary	Cosine similarity	Relevance feedback
Latent Semantic Indexing (LSI)	Probabilistic Inference Network (PIN)	Probabilistic Latent Semantic Indexing (PLSI)	Raw	Dice's coefficient	Thesaurus
	Best Match 25 (BM25)[a]	Latent Dirichlet Allocation (LDA)	Term Frequency Inverse Document Frequency (TFIDF)	Jaccard index	Phrasing
		Correlated Topics Model (CTM)	Best Match 25 (BM25)[a]	Jensen-Shannon divergence (JS)	Clustering
		Relational Topics Model (RTM)			

[a] Okapi BM25 is used to refer both to a non-binary probabilistic model, and its weighting scheme.

Gain (DCG) [246] (a graded relevance scale based on the position of a document among search results). To address this matter in the specific application of trace recovery, Sundaram *et al.* [443] proposed *DiffAR*, *DiffMR*, and *Lag* to assess the quality of retrieved candidate links.

2.2 IR-based Support in a Trace Recovery Process

As the candidate trace links generated by state-of-the-art IR-based trace recovery typically are too inaccurate, the current tools are proposed to be used in a semi-automatic process. De Lucia *et al.* describe this process as a sequence of four key steps, where the fourth step requires human judgement [145]. Although steps 2 and 3 mainly apply to algebraic IR models, also other IR models can be described by a similar sequential process flow. The four steps are:

1. document parsing, extraction, and preprocessing
2. corpus indexing with an IR method
3. ranked list generation

4. analysis of candidate links

In the first step, the artifacts in the targeted information space are processed and represented as a set of documents at a given granularity level, e.g. sections, class files or individual requirements. In the second step, for algebraic IR models, features from the set of documents are extracted and weighted to create an index. When also the query has been indexed in the same way, the output from step 2 is used to calculate similarities between artifacts to rank candidate trace links accordingly. In the final step, these candidate trace links are provided to an engineer for examination. Typically, the engineer then reviews the candidate source and target artifacts of every candidate trace link, and determines whether the link should be confirmed or not. Consequently, the final outcome of the process of IR-based trace recovery is based on human judgment. Concrete examples, put in work task contexts, are presented in Section 3.4.

A number of publications propose advice for engineers working with candidate trace links. De Lucia *et al.* have suggested that an engineer should iteratively decrease the similarity threshold, and stop considering candidate trace links when the fraction of incorrect links get too high [146,147]. Based on an experiment with student subjects, they concluded that an incremental approach in general both improves the accuracy and reduces the effort involved in a tracing task supported by IR-based trace recovery. Furthermore, they report that the subjects preferred working in an incremental manner. Working incrementally with candidate trace links can to some subjects also be an intuitive approach. In a previous experiment by Borg and Pfahl [72], several subjects described such an approach to deal with tool output, even without explicit instructions. Coverage analysis is another strategy proposed by De Lucia *et al.* [149], intended to follow up on the step of iteratively decreasing the similarity threshold. By analyzing the confirmed candidate trace links, i.e. conducting a coverage analysis, De Lucia *et al.* suggest that engineers should focus on tracing artifacts that have few trace links. Also, in an experiment with students, they demonstrated that an engineer working according to this strategy recovers more correct trace links.

3 Related Work

This section presents a chronological overview of IR-based trace recovery, previous overviews of the field, and related work on advancing empirical evaluations of IR-based trace recovery.

3.1 A Brief History of IR-based Trace Recovery

Tool support for the linking process of NL artifacts has been explored by researchers since at least the early 1990s. Pioneering work was done in the LESD project (Linguistic Engineering for Software Design) by Borillo *et al.* [81], in

which a tool suite analyzing NL requirements was developed. The tool suite parsed NL requirements to build semantic representations, and used artificial intelligence approaches to help engineers establish trace links between individual requirements [83]. Apart from analyzing relations between artifacts, the tools evaluated consistency, completeness, verifiability and modifiability [99]. In 1998, a study by Fiutem and Antoniol presented a recovery process to bridge the gap between design and code, based on edit distances between artifacts [183]. They coined the term "*traceability recovery*", and Antoniol *et al.* published several papers on the topic. Also, they were the first to clearly express identification of trace links as an IR problem [16]. Their milestone work from 2002 compared two standard IR models, probabilistic retrieval using the BIM and the VSM [18]. Simultaneously, in the late 1990s, Park *et al.* worked on tracing dependencies between artifacts using a sliding window combined with syntactic parsing. Similarities between sentences were calculated using cosine similarities [369].

During the first decade of the new millennium, several research groups advanced IR-based trace recovery. Natt och Dag *et al.* did research on requirement dependencies in the dynamic environment of market-driven requirements engineering [354]. They developed the tool ReqSimile, implementing trace recovery based on the VSM, and later evaluated it in a controlled experiment [355]. A publication by Marcus *et al.* [326], the second most cited article in the field, constitutes a technical milestone in IR-based trace recovery. They introduced Latent Semantic Indexing (LSI) to recover trace links between source code and NL documentation, a technique that has been used by multiple researchers since. Huffman Hayes *et al.* enhanced VSM retrieval with relevance feedback and introduced secondary performance metrics [233] . From early on, their research had a human-oriented perspective, aimed at supporting V&V activities at NASA using their tool RETRO [232].

De Lucia *et al.* have conducted work focused on empirically evaluating LSI-based trace recovery in their document management system ADAMS [142]. They have advanced the empirical foundation by conducting a series of controlled experiments and case studies with student subjects [143, 144, 148]. Cleland-Huang and colleagues have published several studies on IR-based trace recovery. They introduced probabilistic trace recovery using a PIN-based retrieval model, implemented in their tool Poirot [307]. Much of their work has focused on improving the accuracy of their tool by enhancements such as: applying a thesaurus to deal with synonymy [425], extraction of key phrases [500], and using a project glossary to weight the most important terms higher [500].

Recent work on IR-based trace recovery has, with various results, gone beyond the traditional models for information retrieval. In particular, trace recovery supported by probabilistic topic models has been explored by several researchers. Dekhtyar *et al.*combined several IR models using a voting scheme [155] , including the probabilistic topic model Latent Dirachlet Allocation (LDA). Parvathy *et al.* proposed using the Correlated Topic Model (CTM) [370], and Gethers *et al.*

suggested using Relational Topic Model (RTM) [194]. Abadi *et al.* proposed using Probabilistic Latent Semantic Indexing (PLSI) and utilizing two concepts based on information theory [1], Sufficient Dimensionality Reduction (SDR) and Jensen-Shannon Divergence (JS). Capobianco *et al.* proposed representing NL artifacts as B-splines and calculating similarities as distances between them on the Cartesian plane [96]. Sultanov and Huffman Hayes implemented trace recovery using a swarm technique [442], an approach in which a non-centralized group of non-intelligent self-organized agents perform work that, when combined, enables conclusions to be drawn.

3.2 Previous Overviews on IR-based Trace Recovery

In the beginning of 2012, a textbook on software traceability edited by Cleland-Huang *et al.* was published [113]. Presenting software traceability from several perspectives, the book contains a chapter authored by De Lucia *et al.*, specifically dedicated to IR-based trace recovery [145]. In the chapter, the authors thoroughly present an overview of the field including references to the most important work. Also, the chapter constitutes a good introduction for readers new to the approach, as it describes the basics of IR models. Consequently, the book chapter by De Lucia *et al.* is closely related to our work. However, our work has different characteristics. First, De Lucia *et al.*'s work has more the character of a textbook, including enough background material on IR, as well as examples of applications in context, to introduce readers to the field as a stand-alone piece of work. Our systematic mapping on the other hand, is not intended as an introduction to the field of IR-based trace recovery, but requires extensive pre-understanding. Second, while De Lucia *et al.* report a large set of references to previous work, the method used to identify previous publications is not reported. Our work instead follows the established guidelines for SMs [273], and reports from every phase of the study in a detailed protocol.

Furthermore, basically every publication on IR-based trace recovery contains some information on previous research in the field. Another good example of a summary of the field was provided by De Lucia *et al.* [148]. Even though the summary was not the primary contribution of the publication, they chronologically described the development, presented 15 trace recovery methods and five tool implementations. They compared underlying IR models, enhancing strategies, evaluation methodologies and types of recovered links. However, regarding both methodological rigor and depth of the analysis, it is not a complete SM. De ucia *et al.* have also surveyed proposed approaches to traceability management for impact analysis [140]. They discussed previous work based on a conceptual framework by Bianchi *et al.* [56], consisting of the three traceability dimensions: type of links, source of information to derive links, and their internal representation. Apart from IR-based methods, the survey by De Lucia *et al.* contains both rule-based and data mining-based trace recovery. Also Binkley and Lawrie have

presented a survey of IR-based trace recovery as part of an overview of applications of IR in software engineering [59]. They concluded that the main focus of the research has been to improve the accuracy of candidate links wrt. P-R values, and that LSI has been the most popular IR model. However, they also report that no IR model has been reported as superior for trace recovery. While our work is similar to previous work, our review is more structured and goes deeper with a more narrow scope.

Another set of publications has presented taxonomies on IR techniques in software engineering. In an attempt to harmonize the terminology of the IR applications, Canfora and Cerulo presented a taxonomy of IR models [92]. However, their surveyed IR applications are not explicitly focusing on software engineering. Furthermore, their proposed taxonomy does not cover recent IR models identified in our study, and the subdivision into 'representation' and 'reasoning' poorly serves our intentions. Falessi *et al.* recently published a comprehensive taxonomy of IR techniques available to identify equivalent requirements [175]. They adopted the term variation point from Software Product Line Engineering [377], to stress the fact that an IR solution is a combination of different, often orthogonal, design choices. They consider an IR solution to consist of a combination of algebraic model, term extraction, weighting scheme and similarity metric. Finally, they conducted an empirical study of various combinations and concluded that simple approaches yielded the most accurate results on their dataset. We share their view on variation points, but fail to apply it since our mapping study is limited by what previous publications report on IR-based trace recovery. Also, their proposed taxonomy only covers algebraic IR models, excluding other models (most importantly, the entire family of probabilistic retrieval).

Concept location (a.k.a. feature location) is a research topic that overlaps trace recovery. It can be seen as the first step of a change impact analysis process [328]. Given a concept (or feature) that is to be modified, the initial information need of a developer is to locate the part of the source code where it is embedded. Clearly, this information need could be fulfilled by utilizing IR. However, we distinguish the topics by considering concept location to be more query-oriented [191]. Furthermore, whereas trace recovery typically is evaluated by linking n artifacts to m other artifacts, evaluations of concept location tend to focus on n queries targeting a document set of m source code artifacts (where $n << m$), as for example in the study by Torchiano and Ricca [448]. Also, while it is often argued that trace recovery should retrieve trace links with a high recall, the goal of concept location is mainly to retrieve one single location in the code with high precision. Dit *et al.* recently published a literature review on feature location [159].

3.3 Related Contributions to the Empirical Study of IR-based Trace Recovery

A number of previous publications have aimed at structuring or advancing the research on IR-based trace recovery, and are thus closely related to our study. An early attempt to advance reporting and conducting of empirical experiments was published by Huffman Hayes and Dekhtyar [229]. Their experimental framework describes the four phases: *definition*, *planning*, *realization* and *interpretation*. In addition, they used their framework to characterize previous publications. Unfortunately, the framework has not been applied frequently and the quality of the reporting of empirical evaluations varies greatly [79]. Huffman Hayes *et al.* also presented the distinction between studies of methods (are the tools capable of providing accurate results fast?) and studies of human analysts (how do humans use the tool output?) [231]. Furthermore, they proposed assessing the accuracy of tool output according to quality intervals named 'acceptable', 'good', and 'excellent', based on Huffman Hayes' industrial experience of working with traceability matrices of various qualities. Huffman Hayes *et al.*'s quality levels were defined to represent the effort that would be required by an engineer to vet an entire candidate traceability matrix.

Considering empirical evaluations, we extend the classifications proposed by Huffman Hayes *et al.* [231] by an adapted version of the *Integrated Cognitive Research Framework* by Ingwersen and Järvelin [237]. Their work aimed at extending the de-facto standard of IR evaluation, the *Laboratory Model of IR Evaluation*, developed in the Cranfield tests in the 60s [120], challenged for its unrealistic lack of user involvement [267]. Ingwersen and Järvelin argued that IR is always evaluated in a context, referred to the innermost context as "*the cave of IR evaluation*", and proposed a framework consisting of four integrated contexts (see Fig. 1). We have adapted their framework to a four-level context taxonomy, tailored for IR-based trace recovery, to classify in which contexts previous evaluations have been conducted, see Table 2. Also, we add a dimension of study environments (university, proprietary, and open source environment), as presented in Figure 12 in Section 5. For more information on the context taxonomy, we refer to our original publication [77].

In the field of IR-based trace recovery, the empirical evaluations are termed very differently by different authors. Some call them 'experiments', others 'case studies', and yet others only 'studies'. We use the following definitions, which are established in the field of software engineering.

Case study in software engineering is an empirical enquiry that draws on multiple sources of evidence to investigate one instance (or a small number of instances) of a contemporary software engineering phenomenon within its real-life context, especially when the boundary between phenomenon and context cannot be clearly specified. [413]

Figure 1: The Integrated Cognitive Research Framework by Ingwersen and Järvelin [237], a framework for IR evaluations in context.

Experiment (or controlled experiment) in software engineering is an empirical enquiry that manipulates one factor or variable of the studied setting. Based in randomization, different treatments are applied to or by different subjects, while keeping other variables constant, and measuring the effects on outcome variables. In human-oriented experiments, humans apply different treatments to objects, while in technology-oriented experiments, different technical treatments are applied to different objects. [477]

Empirical evaluations of IR-based trace recovery may be classified as case studies, if they evaluate the use of, e.g. IR-based trace recovery tools in a complex software engineering environment, where it is not clear whether the tool is the main factor or other factors are at play. These are typically level 4 studies in our taxonomy, see Table 2. Human-oriented controlled experiments may evaluate human performance when using two different IR-tools in an artificial (*in vitro*) or well-controlled real (*in vivo*) environment, typically at level 3 of the taxonomy. The stochastical variation is here primarily assumed to be in the human behavior, although there of course are interactions between the human behavior, the artifacts and the tools. Technology-oriented controlled experiments evaluate tool performance on different artifacts, without human intervention, corresponding to levels 1 and 2 in our taxonomy. The variation factor is here the artifacts, and hence the technology-oriented experiment may be seen as benchmarking studies, where one technique is compared to another technique, using the same artifacts, or the performance of one technique is compared for multiple different artifacts.

The validity of the datasets used as input in evaluations in IR-based trace recovery is frequently discussed in the literature. Also, two recent publications primarily address this issue. Ali *et al.* present a literature review on characteristics of artifacts reported to impact trace recovery evaluations [10], e.g. ambiguous and vague requirements, and the quality of source code identifiers. Ali *et al.* extracted P-R values from eight previous trace recovery evaluations, not limited to IR-based trace recovery, and show that the same techniques generate candidate trace links of very different accuracy across datasets. They conclude that research targeting

Table 2: A context taxonomy of IR-based trace recovery evaluations. Level 1 is technology-oriented, and level 3 and 4 are human-oriented. Level 2 typically has a mixed focus.

Level 1: Retrieval context	The most simplified context, referred to as "*the cave of IR evaluation*". A strict retrieval context, performance is evaluated wrt. the accuracy of a set of search results. Quantitative studies dominate.	Precision, recall, F-measure	Experiments on benchmarks, possibly with simulated feedback
Level 2: Seeking context	A first step towards realistic applications of the tool, "drifting outside the cave'. A seeking context with a focus on how the human finds relevant information in what was retrieved by the system. Quantitative studies dominate.	Secondary measures. General IR: MAP, DCG. Traceability specific: Lag, DiffAR, DiffMR.	Experiments on benchmarks, possibly with simulated feedback
Level 3: Work task context	Humans complete real tasks, but in an in-vitro setting. Goal of evaluation is to assess the casual effect of an IR tool when completing a task. A mix of quantitative and qualitative studies.	Time spent on task and quality of work.	Controlled experiments with human subjects.
Level 4: Project context	Evaluations in a social-organizational context. The IR tool is studied when used by engineers within the full complexity of an in-vivo setting. Qualitative studies dominate.	User satisfaction, tool usage	Case studies

only recovery methods in isolation is not expected to lead to any major breakthroughs, instead they suggest that factors impacting the input artifacts should be better controlled. Borg *et al.* recently highlighted that a majority of previous evaluations of IR-based trace recovery have been conducted using artifacts developed by students [79]. The authors explored this potential validity threat in a survey of the traceability community. Their results indicate that while most authors consider artifacts originating from student projects to be only partly representative to industrial artifacts, few respondents explicitly validated them before using them as experimental input.

3.4 Precision and Recall Evaluation Styles for Technology-oriented Trace Recovery

In the primary publications, two principally different styles to report output from technology-oriented experiments have been used, i.e. presentation of P-R values from evaluations in the retrieval and seeking contexts. A number of publications, including the pioneering work by Antoniol *et al.* [18], used the traditional style from the ad hoc retrieval task organized by the Text REtrieval Conference (TREC) [465], driving large-scale evaluations of IR. In this style, a number of queries are executed on a document set, and each query results in a ranked list of

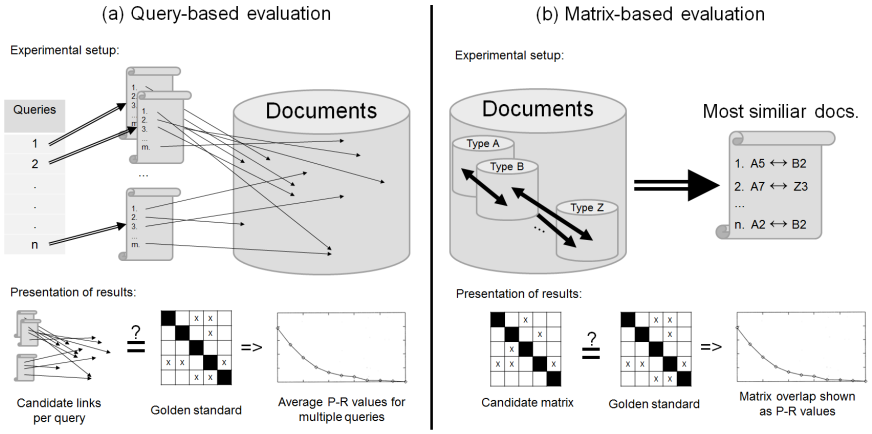


Figure 2: Query-based evaluation vs. matrix-based evaluation of IR-based trace recovery.

search results (cf. (a) in Fig. 2). The accuracy of the IR system is then calculated as an average of the precision and recall over the queries. For example, in Antoniol *et al.*'s evaluation, source code files were used as queries and the document set consisted of individual manual pages. We refer to this reporting style as *query-based* evaluation. This setup evaluates the IR problem: "given this trace artifact, to which other trace artifacts should trace links be established?" The IR problem is reformulated for each trace artifact used as a query, and the results can be presented as a P-R curve displaying the average accuracy of candidate trace links over n queries. This reporting style shows how accurately an IR-based trace recovery tool supports a work task that requires single on-demand tracing efforts (a.k.a. reactive tracing or just-in-time tracing), e.g. establishing traces as part of an impact analysis work task [18, 72, 304].

In the other type of reporting style used in the primary publications, documents of different types are compared to each other, and the result from the similarity- or probability-based retrieval is reported as one single ranked list of candidate trace links. This can be interpreted as the IR problem: "among all these possible trace links, which trace links should be established?" Thus, the outcome is a candidate traceability matrix. We refer to this reporting style as *matrix-based* evaluation. The candidate traceability matrix can be compared to a gold standard, and the accuracy (i.e. overlap between the matrices) can be presented as a P-R curve, as shown in b) in Figure 2. This evaluation setup has been used in several primary publications to assess the accuracy of candidate traceability matrices generated by IR-based trace recovery tools. Also, Huffman Hayes *et al.* defined the quality intervals described in Section 3.3 to support this evaluation style [231].

Apart from the principally different meaning of reported P-R values, the pri-

mary publications also differ by which sets of P-R values are reported. Precision and recall are set-based measures, and the accuracy of a set of candidate trace links (or candidate trace matrix) depends on which links are considered the tool output. Apart from the traditional way of reporting precision at fixed levels of recall, further described in Section 4.3, different strategies for selecting subsets of candidate trace links have been proposed. Such heuristics can be used by engineers working with IR-based trace recovery tools, and several primary publications report corresponding P-R values. We refer to these different approaches to consider subsets of ranked candidate trace links as *cut-off strategies*. Example cut-off strategies include: *Constant cut point*, a fixed number of the top-ranked trace links are selected, e.g. 5, 10, or 50. *Variable cut point*, a fixed percentage of the total number of candidate trace links is selected, e.g. 5% or 10%. *Constant threshold*, all candidate trace links representing similarities (or probabilities) above a specified threshold is selected, e.g. above a cosine similarity of 0.7. *Variable threshold*, a new similarity interval is defined by the minimum and maximum similarities (i.e. similarities are normalized against the highest and lowest similarities), and either a percentage of the candidate trace links are selected or a new constant threshold is introduced.

The choice of what subset of candidate trace links to represent by P-R values reflects the cut-off strategy an imagined engineer could use when working with the tool output. However, which strategy results in the most accurate subset of trace links depends on the specific case evaluated. Moreover, in reality it is possible that engineers would not be consistent in how they work with candidate trace links. As a consequence of the many possibly ways to report P-R values, the primary publications view output from IR-based trace recovery tools from rather different perspectives. For work tasks supported by a separate list of candidate trace links per source artifact, there are indications that human subjects seldom consider more than 10 candidate trace links [72], in line with what is commonplace to present as a 'pages-worth' output of major search engines such as Google, Bing and Yahoo. On the other hand, when an IR-based trace recovery tool is used to generate a candidate traceability matrix over an entire information space, considering only the first 10 candidate links would obviously be insufficient, as there would likely be thousands of correct trace links to recover. However, regardless of reporting style, the number of candidate trace links a P-R value represents is important in any evaluation of IR-based trace recovery tools, since a human is intended to vet the output.

The two inherently different use cases of an IR-based trace recovery tool, reflected by the split into matrix-based and query-based evaluations, also call for different evaluations regarding cut-off strategies. The first main use case of an IR-based trace recovery tool is when one or more candidate trace links from a specific artifact are requested by an engineer. For example, as part of a formal change impact analysis, a software engineer might need to specify which test cases to execute to verify that a defect report has been properly resolved. This example is close to the general definition of IR, "*to find documents that satisfy an information need*

from within large collections". If the database of test cases contains overlapping test cases, it is possible that the engineer needs to report just one suitable test case. In this case precision is more important than recall, and it is fundamental that the tool presents few false positives among the top candidate trace links. Evaluating the performance of the IR-based trace recovery tool using constant cut points is suitable.

The second main use case of an IR-based trace recovery tool is to generate an entire set of trace links, i.e. a candidate traceability matrix. For instance, a traceability matrix needs to be established between n functional requirements and m test case descriptions during software evolution of a legacy system. If the number of artifacts is $n + m$, the number of possible trace links (i.e. the number of pairwise comparisons needed) to consider is $n * m$, a number that quickly becomes infeasible for manual work. An engineer can in such cases use the output from an IR-based trace recovery tool as a starting point. The generation of a traceability matrix corresponds to running multiple simultaneous queries in a general IR system, and typically recall is favored over precision. There is no natural equivalent to this use case in the general IR domain. Furthermore, when generating an entire traceability matrix, it is improbable that the total number of correct trace links is known *a priori*, and consequently constant cut-points are less meaningful. A naïve cut-off strategy is to instead simply use a constant similarity threshold such as the cosine similarity 0.7. More promising cut-off strategies are based on variable thresholds or incremental approaches, as described in Section 2.2. Typically, the accuracies of traceability matrices generated by IR-based trace recovery tools are evaluated *a posteriori*, by analyzing how precision varies for different levels of recall.

4 Method

The overall goal of this study was to form a comprehensive overview of the existing research on IR-based trace recovery. To achieve this objective, we systematically collected empirical evidence to answer research questions characteristic for an SM [273, 373]. The study was conducted in the following distinct steps, (i) development of the review protocol, (ii) selection of publications, (iii) data extraction and mapping of publications, which were partly iterated and each of them was validated.

4.1 Protocol Development

Following the established guidelines for secondary studies in software engineering [273], we iteratively developed a review protocol in consensus meetings between the authors. The protocol defined the research questions (stated in Section 1), the search strategy (described in Section 4.2), the inclusion/exclusion criteria (presented in Table 3), and the classification scheme used for the data extraction

Table 3: Inclusion/exclusion criteria applied in our study. The rightmost column motivates our decisions.

	Inclusion criteria	**Rationale/comments**
I1	Publication available in English in full text	We assumed that all relevant publications would be available in English.
I2	Publication is a peer-reviewed piece of software engineering work	As a quality assurance, we did not include technical reports, master theses etc.
I3	Publication contains empirical results (case study, experiment, survey etc.) of IR-based trace recovery where natural language artifacts are either source or target	Defined our main scope based on our RQs. Publication should clearly link artifacts, thus we excluded tools supporting a broader sense of program understanding such as CONCONUT [138]. Also, the approach should treat the linking as an IR problem. However, we excluded solutions exclusively extracting specific character sequences in NL text, such as work on Mozilla defect reports [33].
	Exclusion criteria	**Rationale/comments**
E1	Answer is no to I1, I2 or I3	
E2	Publication proposes one of the following approaches to recover trace links, rather than IR: a) rule-based extraction b) ontology-based extraction c) machine learning approaches that require supervised learning d) dynamic/execution analysis	We included only publications that are deployable in an industrial setting with limited effort. Thus, we limited our study to techniques that require nothing but unstructured NL text as input. Other approaches could arguably be applied to perform IR, but are too different to fit our scope. Excluded approaches include: rules [168, 437], ontologies [29], supervised machine learning [436], semantic networks [310], and dynamic analysis [169].
E3	Article explicitly targets one of the following topics, instead of trace recovery: a) concept/feature location b) duplicate/clone detection c) code clustering d) class cohesion e) cross cutting concerns/aspect mining	We excluded both concept location and duplicate detection since it deals with different problems, even if some studies apply IR models. Excluded publications include: duplicate detection of defects [410], detection of equivalent requirements [175], and concept location [328]. We explicitly added the topics code clustering, class cohesion, and cross cutting concerns to clarify our scope.

(described in Section 4.3). The extracted data were organized in a tabular format to support comparison across studies. Evidence was summarized per category, and commonalities and differences between studies were investigated. Also, the review protocol specified the use of Zotero[2] as the reference management system, to simplify general tasks such as sorting, searching and removal of duplicates. An important deviation from the terminology used in the guidelines is that we distinguish between *primary publications* (i.e. included units of publication) and *primary studies* (i.e. included pieces of empirical evidence), since a number of publications report multiple studies.

Table 3 states our inclusion/exclusion criteria, along with rationales and examples. A number of general decisions accompanied the criteria:

- Empirical results presented in several articles, we only included from the most extensive publication. Examples of excluded publications include pi-

[2] www.zotero.org

oneering work later extended to journal publications, the most notable being work by Antoniol *et al.* [16] and Marcus and Maletic [326]. However, we included publications describing all *independent replications* (deliberate variations of one or more major aspects), and *dependent replications* (same or very similar experimental setups) by other researchers [429].

- Our study included publications that apply techniques in E2 a–d in Table 3, but use an IR model as benchmark. In such cases, we included the IR benchmark, and noted possible complementary approaches as enhancements. An example is work using probabilistic retrieval enhanced by machine learning from existing trace links [157].
- We included approaches that use structured NL as input, i.e. source code or tabular data, but treat the information as unstructured. Instead, we considered any attempts to utilize document structure as enhancements.
- Our study only included linking between software artifacts, i.e. artifacts that are produced and maintained during development [284]. Thus, we excluded linking approaches to entities such as e-mails [35] and tacit knowledge [228, 439].
- We excluded studies evaluating trace recovery in which neither the source nor the target artifacts dominantly represent information as NL text. Excluded publications comprise linking source code to test code [461], and work linking source code to text expressed in specific modelling notation [20, 117].

4.2 Selection of Publications

The systematic identification of publications consisted of two main phases: (i) development of a gold standard of primary publications, and (ii) a search string that retrieves them, and a systematic search for publications, as shown in Figure 3. In the first phase, a set of publications was identified through exploratory searching, mainly by snowball sampling from a subset of an informal literature review. The most frequently recurring publication fora were then scanned for additional publications. This activity resulted in 59 publications, which was deemed our gold standard[3]. The first phase led to an understanding of the terminology used in the field, and made it possible to develop valid search terms.

The second step of the first phase consisted of iterative development of the search string. Together with a librarian at the department, we repeatedly evaluated our search string using combined searches in the Inspec/Compendex databases. Fifty-five papers in the gold standard were available in those databases. We considered the search string good enough when it resulted in 224 unique hits with

[3]The gold standard was not considered the end goal of our study, but was the target during the iterative development of the search string described next.

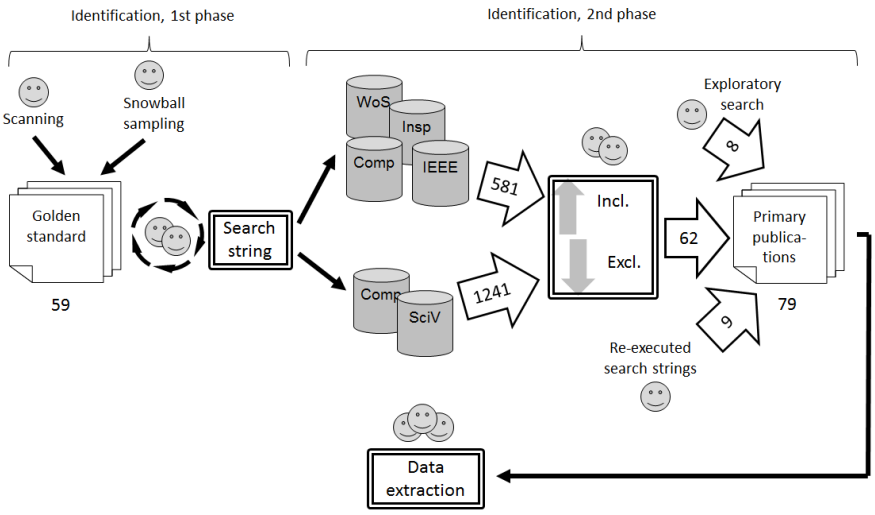


Figure 3: Overview of the publication selection phase. Smileys show the number of people involved in a step, while double frames represent a validation. Numbers refer to number of publications.

80% recall and 20% precision when searching for the gold standard, i.e. 44 of the 55 primary publications plus 176 additional publications were retrieved.

The final search string was composed of four parts connected with ANDs, specifying the *activity*, *objects*, *domain*, and *approach* respectively.

```
(traceability OR "requirements tracing" OR "requirements trace" OR
 "trace retrieval")
AND
(requirement* OR specification* OR document OR documents OR
 design OR code OR test OR tests OR defect* OR artefact* OR
 artifact* OR link OR links)
AND
(software OR program OR source OR analyst)
AND
("information retrieval" OR IR OR linguistic OR lexical OR
 semantic OR NLP OR recovery OR retrieval)
```

The search string was first applied to the four databases supporting export of search results to BibTeX format, as presented in Table 4. The resulting 581 papers were merged in Zotero. After manual removal of duplicates, 281 unique publications remained. This result equals 91% recall and 18% precision compared to the gold standard. The publications were filtered by our inclusion/exclusion criteria, as shown in Table 3, and specified in Section 4.1. Borderline articles were discussed in a joint session of the first two authors. Our inclusion/exclusion criteria

Table 4: Search options used in databases, and the number of search results.

Primary Databases	**Search options**	**#Search results**
Inspec	Title+abstract, no auto-stem	194
Compendex	Title+abstract, no auto-stem	143
IEEE Explore	All fields	136
Web of Science	Title+abstract+keywords	108
Secondary Databases	**Search options**	**#Search results**
ACM Digital Library	All fields, auto-stem	1038
SciVerse Hub Beta	Science Direct+SCOPUS	203

were validated by having the last two authors compare 10% of the 581 papers retrieved from the primary databases. The comparison resulted in a free-marginal multi-rater kappa of 0.85 [387], which constitutes a substantial inter-rater agreement.

As the next step, we applied the search string to two databases without BibTeX export support. One of them, ACM Digital Library, automatically stemmed the search terms, resulting in more than 1000 search results. The inclusion/exclusion criteria were then applied to the total 1241 publications. This step extended our primary studies by 13 publications, after duplicate removal, and application of inclusion/exclusion criteria, 10 identified in ACM Digital Library and 3 from SciVerse.

As the last step of our publication selection phase, we again conducted exploratory searching. Based on our new understanding of the domain, we scanned the top publication fora and the most published scholars for missed publications. As a last complement, we searched for publications using Google Scholar. In total, this last phase identified 8 additional publications. Thus, the systematic database search generated 89% of the total number of primary publications, which is in accordance with expectations from the validation of the search string.

As a final validation step, we visualized the selection of the 70 primary publications using REVIS, a tool developed to support SLRs based on visual text mining [180]. REVIS takes a set of primary publications in an extended BibTeX format and, as presented in Figure 4, visualizes the set as a document map (a), edge bundles (b), and a citation network for the document set (c). While REVIS was developed to support the entire mapping process, we solely used the tool as a means to visually validate our selection of publications.

In Figure 4, every node represents a publication, and a black outline distinguishes primary publications (in c), not only primary publications are visualized). In a), the document map, similarity of the language used in title and abstract is presented, calculated using the VSM and cosine similarities. In the clustering, only absolute distances between publications carry a meaning. The arrows point out Antoniol *et al.*'s publication from 2002 [18], the most cited publication on IR-

based trace recovery. The closest publications in a) are also authored by Antoniol *et al.* [17, 19]. An analysis of a) showed that publications sharing many co-authors tend to congregate. As an example, all primary publications authored by De Lucia *et al.* [141–144, 146–149], Capobianco *et al.* [95, 96], and Oliveto *et al.* [360] are found within the rectangle. No single outlier stands out, indicating that none of the primary publications uses a very different language.

In b), the internal reference structure of the primary studies is shown, displayed by edges connecting primary publications in the outer circle. Analyzing the citations between the primary publications shows one outlier, just below the arrow. The publication by Park *et al.* [369], describing work conducted concurrently with Antoniol *et al.* [18], has not been cited by any primary publications. This questioned the inclusion of the work by Park *et al.*, but as it meets our inclusion/exclusion criteria described in Section 4.1, we decided to keep it.

Finally, in c), the total citation network of the primary studies is presented. Regarding common citations in total, again Park *et al.* is an outlier [369], shown as I in c). The two other salient data points, II and III, are both authored by Natt och Dag *et al.* [352, 355]. However, according to our inclusion/exclusion criteria, there is no doubt that they should be among the primary publications. Thus, in December 2011, we concluded the set of 70 primary publications.

However, as IR-based trace recovery is an active research field, several new studies were published while this publication was in submission. To catch up with the latest research, we re-executed the search string in the databases listed in Table 4 in June 2012, to catch up with publications from the second half of 2011. This step resulted in 9 additional publications, increasing the number of primary publications to 79. In the rest of this paper, we refer to the original 70 publications as the "core primary publications", and the 79 publications as just the "primary publications".

4.3 Data Extraction and Mapping

During the stage of the study, data was extracted from the primary publications according to the pre-defined extraction form of the review protocol. We extracted general information (title, authors, affiliation, publication forum, citations), details about the applied IR approach (IR model applied, selection and weighting of features, enhancements) and information about the empirical evaluation (types of artifacts linked, size and origin of dataset, research methodology, context of IR evaluation, results of evaluation).

The extraction process was validated by the second and third authors, working on a 30% sample of the core primary publications. Half the sample, 15% of the core primary publications, was used to validate extraction of IR details. The other half was used by the other author to validate empirical details. As expected, the validation process showed that the data extraction activity, and the qualitative analysis inherent in that work, inevitably leads to some deviating interpretations.

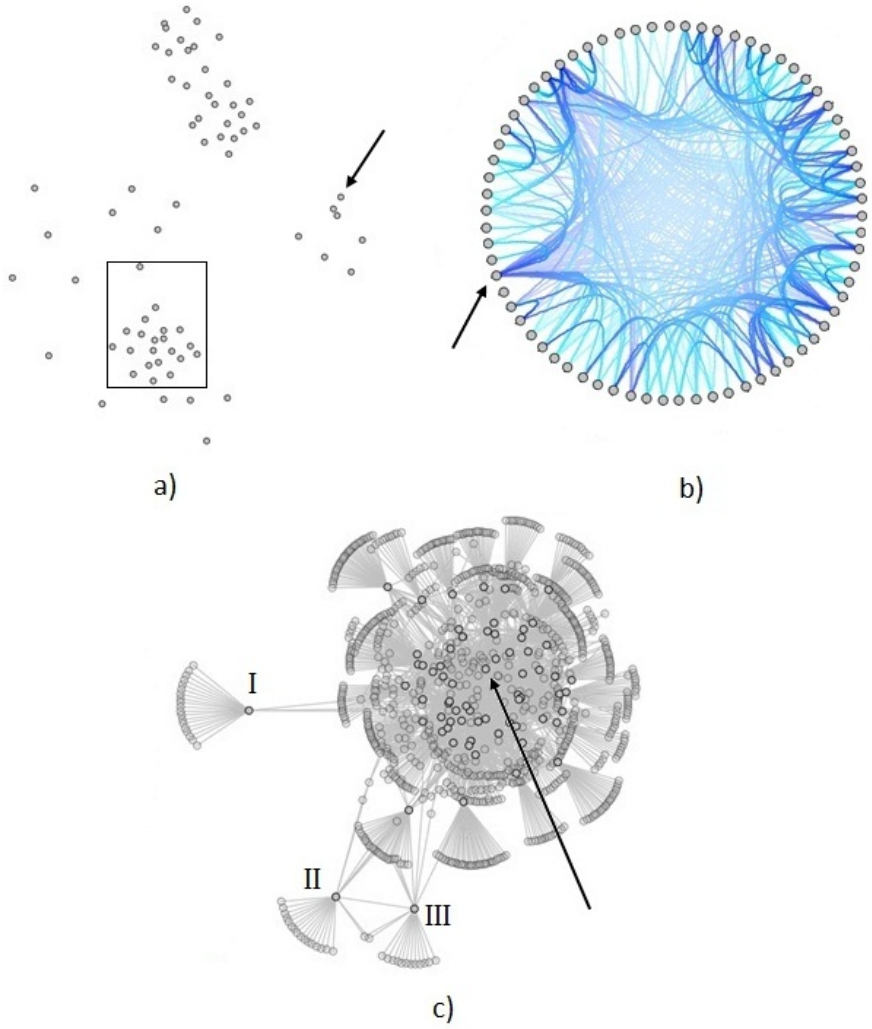


Figure 4: Visualization of core primary publications. a) document map, shows similarities in language among the core primary publications. b) edge bundle, displays citations among the core primary publications. c) citation network, shows shared citations among the core primary publications.

Classifying according to the four levels of IR contexts, which was validated for the entire 30% sample, showed the least consensus. This divergence, and other minor discrepancies detected, were discussed until an agreement was found and followed for the rest of the primary publications. Regarding the IR contexts in particular, we adopted an inclusive strategy, typically selecting the higher levels for borderline publications.

4.4 Threats to Validity

Threats to the validity of the mapping study are analyzed with respect to construct validity, reliability, internal validity and external validity [413]. Particularly, we report deviations from the study guidelines [273].

Construct validity concerns the relation between measures used in the study and the theories in which the research questions are grounded. In this study, this concerns the identification of papers, which is inherently qualitative and dependent on the coherence of the terminology in the field. To mitigate this threat, we took the following actions. The search string we used was validated using a golden set of publications, and we executed it in six different publication databases. Furthermore, our subsequent exploratory search further improved our publication coverage. A single researcher applied the inclusion/exclusion criteria, although, as a validation proposed by Kitchenham and Charters [273], another researcher justified 10% of the search results from the primary databases. There is a risk that the specific terms of the search string related to 'activity' (e.g. "requirements tracing") and 'objects' cause a bias toward both requirements research and publications with technical focus. However, the golden set of publications was established by a broad scanning of related work, using both searching and browsing, and was not restricted to specific search terms.

An important threat to *reliability* concerns whether other researchers would come to the same conclusions based on the publications we selected. The major threat is the extraction of data, as mainly qualitative synthesis was applied, a method that involves interpretation. A single researcher extracted data from the primary publications, and the other two researchers reviewed the process, as suggested by Brereton *et al.* [86]. As a validation, both the reviewers individually repeated the data extraction on a 15% sample of the core primary publications. Another reliability threat is that we present qualitative results with quantitative figures. Thus, the conclusions we draw might depend on the data we decided to visualize; however, the primary studies are publicly available, allowing others to validate our conclusions. Furthermore, as our study contains no formal meta-analysis, no sensitivity analysis was conducted, neither was publication bias explored explicitly.

External validity refers to generalization from this study. In general, the external validity of a SM is strong, as the key idea is to aggregate as much as possible of the available literature. Also, our research scope is tight (cf. the inclusion/ex-

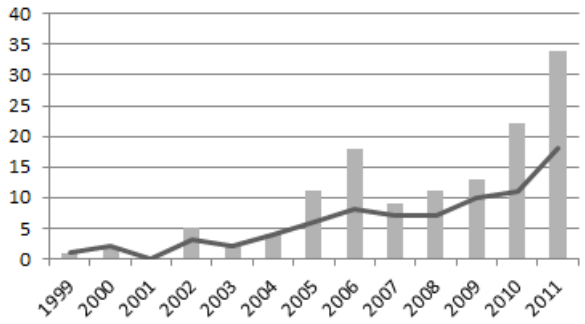


Figure 5: IR-based trace recovery publication trend. The curve shows the number of publications, while the bars display empirical studies in these publications.

clusion criteria in Table 3), and we do not claim that our map applies to other applications of IR in software engineering. Thus, the threats to external validity are minor. Furthermore, as the review protocol is presented in detail in Section 4, other researchers can judge the trustworthiness of the results in relation to the search strategy, inclusion/exclusion criteria, and the applied data extraction. Finally, *internal validity* concerns confounding factors that can affect the causal relationship between the treatment and the outcome. However, as our mapping study does not investigate casual relationships, and only relies on descriptive statistics, this threat is minimal.

5 Results

Following the method defined in Section 4.2, we identified 79 primary publications. Most of the publications were published in conferences or workshops (67 of 79, 85%), while twelve (15%) were published in scientific journals. Table 5 presents the top publication channels for IR-based trace recovery, showing that it spans several research topics. Figure 5 depicts the number of primary publications per year, starting from Antoniol *et al.*'s pioneering work from 1999. Almost 150 authors have contributed to the 79 primary publications, on average writing 2.2 of the articles. The top five authors have on average authored 14 of the primary publications, and are in total included as authors in 53% of the articles. Thus, a wide variety of researchers have been involved in IR-based trace recovery, but there is a group of a few well-published authors. More details and statistics about the primary publications are available in Appendix I.

Several publications report empirical results from multiple evaluations. Consequently, our mapping includes 132 unique empirical contributions, i.e. the mapping comprises results from 132 unique combinations of an applied IR model and

Table 5: Top publication channels for IR-based trace recovery.

Publication forum	#Publications
International Requirements Engineering Conference	9
International Conference on Automated Software Engineering	7
International Conference on Program Comprehension	6
International Workshop on Traceability in Emerging Forms of Software Engineering	6
Working Conference on Reverse Engineering	5
Empirical Software Engineering	4
International Conference on Software Engineering	4
International Conference on Software Maintenance	4
Other publication fora (two or fewer publications)	34

its corresponding evaluation on a dataset. As described in Section 4.1, we denote such a unit of empirical evidence a 'study', to distinguish from 'publications'.

5.1 IR Models Applied to Trace Recovery (RQ1)

In Figure 6, reported studies in the primary publications are mapped according to the (one or several) IR models applied, as defined in Section 2. The most frequently reported IR models are the algebraic models, VSM and LSI. For LSI, the dimensionality reductions applied in previous studies is reported in Appendix I. Various probabilistic models have been applied in 29 of the 132 evaluations, including 14 applications of statistical LMs. Five of the applied approaches do not fit in the taxonomy; examples include utilizing swarm techniques [442] and B-splines [96]. As shown in Figure 6, VSM is the most applied model 2008-2011, however repeatedly as a benchmark to compare new IR models against. An apparent trend is that trace recovery based on LMs has received an increasing research interest during the last years.

Only 47 (72%) of the 65 primary publications with technical foci report which preprocessing operations were applied to NL text. Also, in several publications one might suspect that the complete preprocessing was not reported (e.g. [105] and [282]), possibly due to page restriction. As a result, a reliable report of feature selection for IR-based trace recovery is not possible. Furthermore, several papers do not report any differences regarding preprocessing of NL text and source code (on the other hand some papers make a clear distinction, e.g. [467]). Among the publications reporting preprocessing, 32 report conducting stop word removal

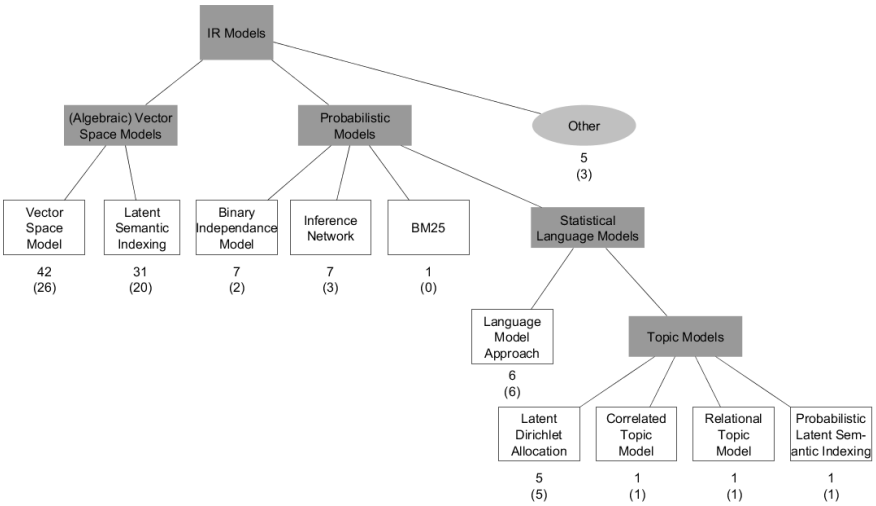


Figure 6: Taxonomy of IR models in trace recovery. The numbers show in how many of the primary publications a specific model has been applied, the numbers in parentheses show IR models applied since 2008.

and stemming, making it the most common combination. The remaining publications report other combinations of stop word removal, stemming and ID splitting. Also, two publications report applying Google Translate as a preprocessing step to translate NL text to English [234, 304]. Figure 7 presents in how many primary publications different preprocessing steps are explicitly mentioned, both for NL text and source code.

Regarding NL text, most primary publications select all terms that remain after preprocessing as features. However, two publications select only nouns and verbs [494, 496], and one selects only nouns [96]. Also, Capobianco *et al.* have explicitly explored the semantic role of nouns [95]. For the purposes of the mapping of primary publications dealing with source code, a majority unfortunately does not clearly report about the feature selection (i.e. selecting which subset of terms to extract to represent the artifact). Seven publications report that only IDs were selected, while four publications selected both IDs and comments. Three other publications report more advanced feature selection, including function arguments, return types and commit comments [1, 9, 92].

Among the primary publications, the weighting scheme applied to selected features is reported in 58 articles. Although arguably more tangible for algebraic retrieval models, feature weighting is also important in probabilistic retrieval. Moreover, most weighing schemes are actually families of configuration variants [416], but since this level of detail often is omitted in publications on IR-based trace recovery, as also noted by Oliveto [359], we were not able to investigate this further.

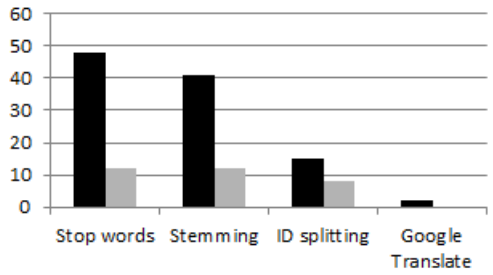


Figure 7: Preprocessing operations used in IR-based trace recovery. The figure shows the number of times a specific operation has been reported in the primary publications. Black bars refer to preprocessing of NL text, gray bars show preprocessing of text extracted from source code.

Figure 8 shows how many times, in the primary publications, various types of feature weighting schemes have been applied. Furthermore, one publication reports upweighting of verbs in the TFIDF weighting scheme, motivated by verbs' nature of describing the functionality of software [321].

Several enhancement strategies to improve the performance of IR-based trace recovery tools are proposed, as presented in Figure 9. The figure shows how many times different enhancement strategies have been applied in the primary publications. Most enhancements aim at improving the precision and recall of the tool output, however also a computation performance enhancement is reported [249]. The most frequently applied enhancement strategy is relevance feedback, applied by e.g. De Lucia *et al.* [146] and Huffman Hayes *et al.* [232], giving the human a chance to judge partial search results, followed by re-executing an improved search query. The following most frequently applied strategies, further described in Section 2.1, are applying a thesaurus to deal with synonyms, (e.g. proposed by Huffman Hayes *et al.* [231] and Leuser and Ott [298]), clustering results based on for instance document structure to improve presentation or ranking of recovered trace links, (explored by e.g. Duan and Cleland-Huang [163] and Zhou and Yu [496]), and phrasing, i.e., going beyond the BoW model by considering sequences of words, e.g. as described by Zou *et al.* [498] and Chen and Grundy [106]. Other enhancement strategies repeatedly applied include: upweighting terms considered important by applying a project glossary, e.g. [499], machine learning approaches to improve results based on for example the existing trace link structure, e.g. [157], and combining the results from different retrieval models in voting systems, e.g. [194]. Yet another set of enhancements have only been proposed in single primary publications, such as query expansion [197], analyses of call graphs [494], regular expressions [106], and smoothing filters [137].

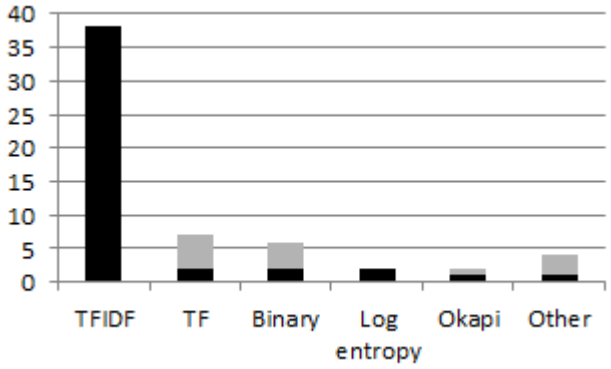


Figure 8: Feature weighting schemes in IR-based trace recovery. Bars depict how many times a specific weighting scheme has been reported in the primary publications. Black color shows reported weighting in publications applying algebraic IR models.

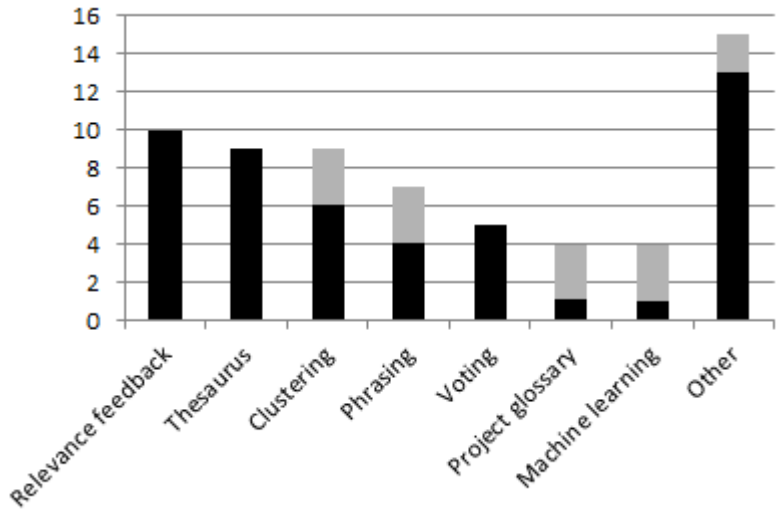


Figure 9: Enhancement strategies in IR-based trace recovery. Bars depict how many times a specific strategy has been reported in the primary publications. Black color represents enhancements reported in publications using algebraic IR models.

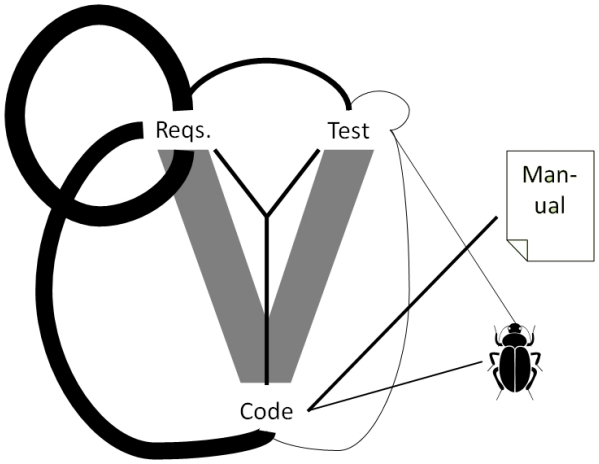
5.2 Types of Software Artifacts Linked (RQ2)

Figure 10 maps onto the classical software development V-model the various software artifact types that are used in IR-based trace recovery evaluations. Requirements, the left part of the model, include all artifacts that specify expectations on a system, e.g. market requirements, system requirements, functional requirements, use cases, and design specifications. The distinction between these are not always possible to derive from the publications, and hence we have grouped them together under the broad label 'requirements'. The right part of the model represents all artifacts related to verification activities, e.g. test case descriptions and test scripts. Source code artifacts constitute the bottom part of the model. Note however, that our inclusion/exclusion criteria, excluding duplication analyses and studies where neither source nor target artifacts are dominated by NL text, results in fewer links between requirements-requirements, code-code, code-test, test-test and defect-defect than would have been the case if we had studied the entire field of IR applications within software engineering.

The most common type of links that has been studied was found to be between requirements (37 evaluations), either of the same type or of different levels of abstraction. The second most studied artifact linking is between requirements and source code (32 evaluations). Then, in decreasing order, mixed links in an information space of requirements, source code and tests (10 evaluations), links between requirements and tests (9 evaluations) and links between source code and manuals (6 evaluations). Less frequently studied trace links include links between source code and defects/change requests (e.g. [193]) and links between tests [316]. In three primary publications, the types of artifacts traced are unclear, either not specified at all or merely described as 'documents' (e.g. [107]).

5.3 Strength of Evidence (RQ3)

An overview of the datasets used for evaluations in the primary publications is shown in Figure 11. In total we identified 132 evaluations; in 42 (32%) cases proprietary artifacts were studied, either originating from development projects in private companies or the US agency NASA. Nineteen (14%) evaluations using artifacts collected from open source projects have been published and 65 (49%) employing artifacts originating from a university environment. Among the datasets from university environments, 34 consist of artifacts developed by students. In six primary publications, the origin of the artifacts is mixed or unclear (e.g., [304,369, 370]. Figure 11 also depicts the sizes of the datasets used in the evaluations, wrt. the number of artifacts. The majority of the evaluations in the primary publications were conducted using an information space of less than 500 artifacts. In 38 of the evaluations, less than 100 artifacts were used as input. The primary publications with the by far highest number of artifacts, evaluated links between 3,779 business requirements and 8,334 market requirements at Baan [352] (now owned by Infor



Reqs.-Reqs.	37
Code-Reqs.	32
Reqs.-Test	10
Reqs.-Code-Test	9
Code-Manual	6
Code-Defect	3
Code-Test	2
Test-Test	1
Test-Defect	1

Figure 10: Types of links recovered in IR-based trace recovery. The table shows the number of times a specific type of link is the recovery target in the primary publications, also represented by the weight of the edges in the figure.

Global Solutions), and trace links between 9 defect reports and 13,380 test cases at Research in Motion [265].

Table 6 presents the six datasets that have been most frequently used in evaluations of IR-based trace recovery, sorted by the number of primary studies in which they were used. CM-1, MODIS, and EasyClinic are publicly available from the CoEST web page[4]. Note that most publicly available datasets except EasyClinic are *bipartite*, i.e., the dataset contains only links between two disjunct subsets of artifacts.

All primary publications report some form of empirical evaluations, a majority (80%) conducting "*studies of methods*" [231]. Fourteen publications (18%) report results regarding the human analyst, two primary publications study both methods and human analysts [18, 143]. Figure 12 shows the primary publications mapped to the four levels of the context taxonomy described in Section 3.3. Note that a number of publications cover more than one environment, due to either mixed artifacts or multiple studies. Also, two publications did not report the environment, and could not be mapped. A majority of the publications (50), exclusively conducted evaluations taking place in the innermost retrieval context, the so-called "*cave of IR evaluation*" [237]. As mentioned in Section 2, evaluations in the cave display an inconsistent use of terminology. Nineteen (38%) of the primary publications refer to their evaluations in the retrieval context as experiments, 22 (44%) call them case studies, and in nine (18%) publications they are merely referred to

[4] www.coest.org

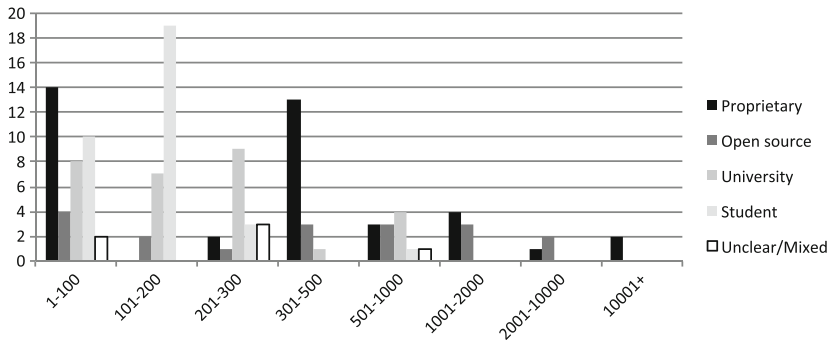


Figure 11: Datasets used in studies on IR-based trace recovery. Bars show number and origin of artifacts.

Table 6: Summary of the datasets most frequently used for evaluations.

#	Dataset	Artifacts	Links	Origin	Development characteristics	Size[a]	Lang.
17	CM-1	Requirements specifying system requirements and detailed design	Bipartite dataset, many-to-many links	NASA	Embedded software development in governmental agency	455	English
16	EasyClinic	Use cases, sequence diagrams, source code, test case descriptions	Many-to-many links	Univ. of Salerno	Student project	150	Italian
8	MODIS	Requirements specifying system requirements and detailed design	Bipartite dataset, many-to-many links.	NASA	Embedded software development in governmental agency	68	English
7	Ice-Breaker System (IBS)	Functional requirements and source code	Not publicly available in full detail	Robertson and Robertson [399]	Textbook on requirements engineering	185	English
6	LEDA	Source code and user documentation	Bipartite dataset, many-to-one links	Max Planck Inst. for Informatics Saarbrücken	Scientific computing	296	English
5	Event-Based Traceability (EBT)	Functional requirements and source code	Not publicly available	DePaul Univ.	Tool from research project	138	English

[a] Size is presented as the total number of artifacts.

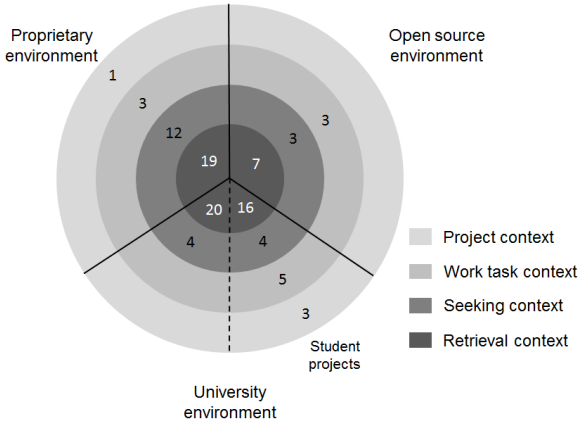


Figure 12: Contexts of evaluations of IR-based trace recovery, along with study environments. Numbers show the number of primary publications that target each combination.

as studies.

Since secondary measures were applied, fourteen publications (18%) are considered to have been conducted in the seeking context. Eleven primary publications conducted evaluations in the work context, mostly through controlled experiments with student subjects. Only three evaluations are reported in the outermost context of IR evaluation, the project context, i.e. evaluating the usefulness of trace recovery in an actual end user environment. Among these, only a single publication reports an evaluation from a non-student development project [304].

6 Discussion

This section discusses the results reported in the previous section and concludes on the research questions. Along with the discussions, we conclude every question with concrete suggestions on how to advance research on IR-based trace recovery. Finally, in Section 6.4, we map our recommendations to the traceability challenges articulated by CoEST [202].

6.1 IR Models Applied to Trace Recovery (RQ1)

During the last decade, a wide variety of IR models have been applied to recover trace links between artifacts. Our study shows that the most frequently applied models have been algebraic, i.e. Salton's classic VSM from the 60s [417] and LSI, the extension developed by Deerswester in the 90s [150]. Also, we show that VSM has been implemented more frequently than LSI, in contrast to what was

reported by Binkley and Lawrie [59]. The interest in algebraic models might have been caused by the straightforwardness of the techniques; they have concrete geometrical interpretations, and are rather easy to understand also for non-IR experts. Moreover, several open source implementations are available. Consequently, the algebraic models are highly applicable to trace recovery studies, and they constitute feasible benchmarks when developing new methods. However, in line with the development in the general IR field [490], LMs [378] have been getting more attention in the last years. Regarding enhancements strategies, relevance feedback, introduction of a thesaurus and clustering of results are the most frequently applied.

While implementing an IR model, the developers inevitably have to make a variety of design decisions. Consequently, this applies also to IR-based trace recovery tools. As a result, tools implementing the same IR model can produce rather different output [77]. Thus, omitting details in the reporting obstructs replications and the possibility to advance the field of trace recovery through secondary studies and evidence-based software engineering techniques [247]. Unfortunately, even fundamental information about the implementation of IR is commonly left out in trace recovery publications. Concrete examples include feature selection and weighting (particularly neglected for publications indexing source code) and the number of dimensions of the LSI subspace. Furthermore, the heterogeneous use of terminology is an unnecessary difficulty in IR-based trace recovery publications. Concerning general traceability terminology, improvements can be expected as Cleland-Huang *et al.* [113] dedicated an entire chapter of their recent book to this issue. However, we hope that Section 2.1 of this paper is a step toward aligning also the IR terminology in the community.

To support future replications and secondary studies on IR-based trace recovery, we suggest that:

- Studies on IR-based trace recovery should use IR terminology consistently, e.g. as presented in Table 1 and Figure 6, and use general traceability terminology as proposed by Cleland-Huang *et al.* [113].
- Authors of articles on IR-based trace recovery should carefully report the implemented IR model, including the features considered, to enable aggregating empirical evidence.
- Technology-oriented experiments on IR-based trace recovery should adhere to rigorous methodologies such as the evaluation framework by Huffman Hayes and Dekhtyar [229].

6.2 Types of Software Artifacts Linked (RQ2)

Most published evaluations on IR-based trace recovery aim at establishing trace links between requirements in a wide sense, or between requirements and source

code. Apparently, the artifacts of the V&V side of the V-model are not as frequently in focus of researchers working on IR-based trace recovery. One can think of several reasons for this unbalance. First, researchers might consider that the structure of the document subspace of the requirement side of the V-model is more important to study, as it is considered the "starting point" of development. Second, the early public availability of a few datasets containing requirements of various kinds, might have paved the way for a series of studies by various researchers. Third, publicly available artifacts from the open source community might contain more requirements artifacts than V&V artifacts. Nevertheless, research on trace recovery would benefit from studies on a more diverse mix of artifacts. For instance, the gap between requirements artifacts and V&V artifacts is an important industrial challenge [414]. Hence, exploring whether IR-based trace recovery could be a way to align "the two ends of software development" is worth an effort.

Apart from the finding that requirement-centric studies on IR-based trace recovery are over-represented, we found that too few studies go beyond trace recovery in bipartite traceability graphs. Such simplified datasets hardly represent the diverse information landscapes of large-scale software development projects. Exceptions include studies by De Lucia *et al.*, who repeatedly have evaluated IR-based trace recovery among use cases, functional requirements, source code and test cases [137, 141, 143, 146–149], however originating from student projects, which reduces the industrial relevance.

To further advance the research of IR-based trace recovery, we suggest that:

- Studies should be conducted on diverse datasets containing a higher number of artifacts, to explore recovery of different types of trace links.
- Studies should go beyond bipartite datasets to better represent the heterogeneous information landscape of software engineering, thus enabling studies on several types of links within the same datasets.

6.3 Strength of Evidence (RQ3)

Most evaluations on IR-based trace recovery were conducted on bipartite datasets containing fewer than 500 artifacts. Obviously, as pointed out by several researchers, any software development project involves much larger information landscapes, that also consist of heterogeneous artifacts. A majority of the evaluations of datasets containing more than 1,000 artifacts were conducted using open source artifacts, an environment in which fewer types of artifacts are typically maintained [92, 419], thus links to or from source code are more likely to be studied. Even though small datasets might be reasonable to study, only two primary publications report from evaluations containing more than 10,000 artifacts [265, 352]. As a result, the question of whether the state-of-the-art IR-based trace recovery scales to larger document spaces or not remains unanswered. In the

empirical NLP community, Banko and Brill [37] showed that some conclusions (related to machine learning techniques for NL disambiguation) drawn on small datasets may not carry over to very large datasets. Researchers on IR-based trace recovery appear to be aware of the scalability issue however, as it is commonly mentioned as a threat to external validity and suggested as future work in the primary publications [144, 197, 233, 297, 322, 467]. On the other hand, one reason for the many studies on small datasets is the challenge involved in obtaining the complete set of correct trace links, i.e. a gold standard or ground truth, required for evaluations. In certain domains, e.g. development of safety-critical systems, such information might already be available. If such information is missing however, a traceability researcher first needs to establish the gold standard, which requires much work for a large dataset.

Regarding the validity of datasets used in evaluations, a majority used artifacts originating from university environments as input. Furthermore, most studies on proprietary artifacts used only the CM-1 or MODIS datasets collected from NASA projects, resulting in their roles as de-facto benchmarks from an industrial context. Clearly, again the external validity of state-of-the-art trace recovery must be questioned. On one hand, benchmarking can be a way to advance IR tool development, as TREC have demonstrated in the general IR research [434], but on the other hand it can also lead the research community to over-engineering tools on specific datasets [77]. Thus, the community needs to consider the risk of optimization against those specific benchmarks, which may make the final result less suitable in the general case, if the benchmarks are not representative enough. The benchmark discussion has been very active in the traceability community the last years [47, 112, 152, 153, 202].

A related problem, in particular for proprietary datasets that cannot be disclosed, is that datasets often are poorly described [79]. In some particular publications, NL artifacts in datasets are only described as 'documents'. Thus, as already discussed related to RQ1 in Section 6.1, inadequate reporting obstructs replications and secondary studies. Moreover, providing information about the datasets and their contexts is also important for interpreting results and their validity, in line with previous work by Ali *et al.* [10] and Borg *et al.* [79]. For example, researchers should report as much of the industrial context from which the dataset arose as encouraged by Kitchenham and Charters [275]. As a starting point, researchers could use the preliminary framework for describing industrial context by Petersen and Wohlin [374].

As discussed in Section 3.4, P-R values can be reported from IR-based trace recovery evaluations in different ways. Unfortunately, the reported values are not always properly explained in the primary publications. In the evaluation report, it is central to state whether a query-based or matrix-based evaluation style has been used, as well as which cut-off strategies were applied. Furthermore, for query-based evaluations (closer resembling traditional IR), we agree with the opinion of Spärck Jones *et al.* [438], that reporting only precision at standard recall levels is

opaque. The figures obscure the actual numbers of retrieved documents needed to get beyond low recall, and should be complemented by P-R values from a constant cut-point cut-off strategy. Moreover, regarding both query-based and matrix-based evaluation styles, reporting also secondary measures (such as MAP and DCG) is a step toward more mature evaluations.

Most empirical evaluations of IR-based trace recovery were conducted in the innermost of IR contexts, i.e. a clear majority of the research was conducted "in the cave" or just outside [237]. For some datasets, the output accuracy of IR models has been well-studied during the last decade. However, more studies on how humans interact with the tools are required; similar to what has been explored by Huffman Hayes *et al.* [128, 154, 230, 233] and De Lucia *et al.* [146–148]. Thus, more evaluations in a work task context or a project context are needed. Regarding the outermost IR context, only one industrial in-vivo evaluation [304] and three evaluations in student projects [142–144] have been reported. Finally, regarding the innermost IR contexts, the discrepancy of methodological terminology should be harmonized in future studies.

To further advance evaluations of IR-based trace recovery, we suggest that:

- The community should continue its struggle to acquire a set of more representative benchmarks.
- Researchers should better characterize both the context and the datasets used in evaluations, in particular when they cannot be disclosed for confidentiality reasons.
- P-R values should be complemented by secondary measures such as MAP and DCG, and it should be made clear whether a query-based or matrix-based evaluation style was used.
- Focus on tool enhancements "in the cave" should be shifted towards evaluations in the work task or project context.

6.4 In the Light of the CoEST Research Agenda

Gotel *et al.* [202] recently published a framework of challenges in traceability research, a CoEST community effort based on a draft from 2006 [116]. The intention of the framework is to provide a structure to direct future research on traceability. CoEST defines eight research themes, addressing challenges that are envisioned to be solved in 2035, as presented in Table 7. Our work mainly contributes to three of the research themes, *purposed traceability*, *trusted traceability*, and *scalable traceability*. Below, we discuss the three research themes in relation to IR-based trace recovery, based on our empirical findings.

The research theme *purposed traceability* charts the development of a classification scheme for traceability contexts, and a collection of possible stakeholder

Table 7: Traceability research themes defined by CoEST [202]. Ubiquitous traceability is referred to as "*the grand challenge of traceability*", since it requires significant progress in the other research themes

Research theme	Goal to reach by 2035
Purposed traceability	to define and instrument prototypical traceability profiles and patterns
Cost-effective traceability	to perform systematic quality assessment and assurance of the traceability
Configurable traceability	to provide for levels of abstraction and granularity in traceability techniques, methods and tools, facilitated by improved trace visualizations, to handle very large datasets and the longevity of these data
Trusted traceability	to develop cost-benefit models for analyzing stakeholder requirements for traceability and associated solution options at a fine-grained level of detail
Scalable traceability	to use dynamic, heterogeneous and semantically rich traceability information models to guide the definition and provision of traceability
Portable traceability	to agree upon universal policies, standards, and a unified representation or language for expressing traceability concepts
Valued traceability	to raise awareness of the value of traceability, to gain buy-in to education and training, and to get commitment to implementation
Ubiquitous traceability	to provide automation such that traceability is encompassed within broader software and systems engineering processes, and is integral to all tool support

requirements on traceability. Also, a "*Traceability Book of Knowledge*" is planned, including terminology, methods, practices and the like. Furthermore, the research agenda calls for additional empirical studies. Our contribution intensifies CoEST's call for additional industrial case studies, by showing that a majority of IR-based trace recovery studies have been conducted in the "cave of IR evaluation". To guide future empirical studies, we propose an adapted version of the model of IR evaluation contexts by Ingwersen and Järvelin [237], tailored for IR-based trace recovery. Also, we confirm the need for a "*Traceability Book of Knowledge*" and an aligned terminology in the traceability community, as our secondary study was obstructed by language discrepancies.

Trusted traceability comprises research to gain improvements in the quality of creation and maintenance of automatic trace links. Also, the research theme calls for empirical evidence as to the quality of traceability methods and tools with respect to the quality of the trace links. Our work, founded in evidence-based software engineering approaches, aggregated the empirical evidence of IR-based trace recovery until December 2011. Based on this, we provide several advice on how to advance future evaluations.

Finally, the research theme *scalable traceability* calls for the traceability community to obtain and publish large industrial datasets from various domains to enable researchers to investigate scalability of traceability methods. Also this call for research is intensified by our work, as we empirically show that alarmingly few evaluations of IR-based trace recovery have been conducted on industrial datasets of representative sizes.

7 Summary and Future Work

Our review of IR-based trace recovery compares 79 publications containing 132 empirical studies, systematically derived according to established procedures [273]. Our study constitutes the most extensive summary of publications of IR-based trace recovery yet published.

More than ten IR models have been applied to trace recovery (RQ1). More studies have evaluated algebraic IR models (i.e. VSM and LSI) than probabilistic models (e.g. BIM, PIN, LM, LDA). A visible trend is, in line with development in the general field of IR, that the probabilistic subset of statistical language models have received increased attention in recent years. While extracting data from the primary publications, it became clear that the inconsistent use of IR terminology is an issue in the field. In an attempt to homogenize the language, we present structure in the form of a hierarchy of IR models (Fig. 6) and a collection of IR terminology (Table 1).

In the 132 mapped empirical studies, artifacts from the entire development process have been linked (RQ2). The dominant artifact type is requirements at various levels of abstraction, followed by source code. Substantially fewer studies

have been conducted on test artifacts, and only single publications have targeted user manuals and defect reports. Furthermore, a majority of the evaluations of IR-based trace recovery have been made on bipartite datasets, i.e. only trace links between two disjoint sets of artifacts were recovered.

Among the 79 primary publications mapped in our study, we conclude that the heterogeneity of reporting detail obstructs the aggregation of empirical evidence (RQ3). Also, most evaluations have been conducted on small bipartite datasets containing fewer than 500 artifacts, which is a severe threat to external validity. Furthermore, a majority of evaluations have been using artifacts originating from a university environment, or a dataset of proprietary artifacts from NASA. As a result, the two small datasets EasyClinic and CM-1 constitute the de-facto benchmark in IR-based trace recovery. Another validity threat to the applicability of IR-based trace recovery is that a clear majority of the evaluations have been conducted in "the cave of IR evaluation" as reported in Figure 12. Instead, the strongest empirical evidence in favor of IR-based trace recovery tools comes from a set of controlled experiments on student subjects, reporting that tool-supported subjects outperform manual control groups. Thus, we argue that industrial in-vivo evaluations are needed to motivate the feasibility of the approach and further studies on the topic, in which IR-based trace recovery should be studied within the full complexity of an industrial setting. As such, our empirical findings intensify the recent call for additional empirical studies by CoEST [202].

In several primary publications it is not made clear whether a query-based or matrix-based evaluation style has been used. Also, the different reporting styles of P-R values make secondary studies on candidate trace link accuracies challenging. We argue that both the standard measures precision at fixed recall levels and P-R at specific document cut-offs should be reported when applicable, complemented by secondary measures such as MAP and DCG.

As a continuation of this literature study, we intend to publish the extracted data to allow for collaborative editing[5], and for interested readers to review the details. A possible future study would be to conduct a deeper analysis of the enhancement strategies that have been reported as successful in the primary publications, to investigate patterns concerning in which contexts they have been successfully applied. Another option for the future is to aggregate results from the innermost evaluation context, as P-R values repeatedly have been reported in the primary studies. However, such a secondary study must be carefully designed to allow a valid synthesis across different studies. Finally, future work could include other mapping dimensions, such as categorizing the primary publications according to other frameworks, e.g positioning them related to the CoEST research themes.

[5]Made available online since the original paper was printed: http://sites.google.com/site/tracerepo/

Acknowledgement

This work was funded by the Industrial Excellence Center EASE – Embedded Applications Software Engineering[6]. Thanks go to our librarian Mats Berglund for working on the search strings, and Lorand Dali for excellent comments on IR details.

[6] http://ease.cs.lth.se

Appendix I: Classification of Primary Publications

Tables 8-12 present our classification of the primary publications, sorted by number of citations according to Google Scholar (July 1, 2012). Note that the well-cited works by Marcus and Maletic [326] (354 citations) and Antoniol *et al.* [16] (85 citations) are not listed. Applied IR models are reported in the fourth column. For LSI, the number of dimensions (k) in the reduced term-document space is reported in parenthesis, divided per dataset when possible. The number of dimensions is reported either as a fixed number of dimensions, an interval of dimensions, a dimensionality reduction in percent, or 'N/A' when the information is not available. A bold number represents the best choice, as concluded by the original authors. Regarding LDA, the number of topics (t) is reported. Datasets are classified according to origin: proprietary (Ind), open source (OS), university (Univ), student (Stud), not clearly reported (Unclear), and mixed origin (Mixed). Numbers in parentheses show the number of artifacts studied, i.e. the total number of artifacts in the dataset, 'N/A' is used when it is not reported. Unless the full dataset name is presented, the following abbreviations are used: IBS (Ice Breaker System), EBT (Event-Based Traceability), LC (Light Control system), TM (Transient Meter). Evaluation, the rightmost column, maps primary publications to the context taxonomy described in Section 3 (Level 1-4 = retrieval context, seeking context, work task context, project context). Finally, Table 13 shows the distinctly most productive authors and affiliations, based upon our primary publications.

Table 8: Classification of primary publications, part I.

Cit.	Title	Authors	IR mod.	Dataset	Evaluation
486	Recovering Traceability Links between Code and Documentation	Antoniol, Canfora, De Lucia, Merlo	BIM, VSM	Univ: LEDA (296), Stud: Albergate (116)	Level 1, Level 3 (8 subj.)
205	Advancing Candidate Link Tracing: Generation for Requirements The Study of Methods	Huffman Hayes, Dekhtyar, Sundaram	VSM, LSI (k=10 (MODIS), 100 (CM-1))	Ind: MODIS (68), CM-1 (455)	Level 2
169	Improving Requirements Tracing via Information Retrieval	Huffman Hayes, Dekhtyar, Osborne	VSM	Ind: MODIS (68)	Level 1
140	Recovering Traceability Links in Software Artifact Management Systems Using Information Retrieval Methods	De Lucia, Fasano, Oliveto, Tortora	LSI (k=30-100%)	Stud: (Multiple projects)	Level 4 (150 subj.)
99	Utilizing Supporting Evidence to Improve Dynamic Requirements Traceability	Cleland-Huang, Settimi, Duan, Zou	PIN	Univ: IBS (252), EBT (114), LC (61)	Level 1
79	Best Practices for Automated Traceability	Cleland-Huang, Berenbach, Clark, Settimi, Romanova	PIN	Ind: Siemens Logistics and Automation (N/A), Univ: IBT (255), EBT (114)	Level 1
74	Helping Analysts Trace Requirements: An Objective Look	Huffman Hayes, Dekhtyar, Sundaram, Howard	VSM	Ind: MODIS (68)	Level 2
70	Can LSI help Reconstructing Requirements Traceability in Design and Test?	Lormans, van Deursen	LSI (k=20%)	Ind: Philips (359), Stud: PacMan (46), Callisto (N/A)	Level 1
68	Supporting Software Evolution through Dynamically Retrieving Traces to UML Artifacts	Settimi, Cleland-Huang, BenKhadra, Mody, Lukasik, DePalma	VSM	Univ: EBT (138)	Level 1
64	Enhancing an Artefact Management System with Traceability Recovery Features	De Lucia, Fasano, Oliveto, Tortora	LSI (k=10-50%)	Stud: EasyClinic (150)	Level 1

Table 9: Classification of primary publications, part II.

Cit.	Title	Authors	IR mod.	Dataset	Evaluation
58	Recovery of Traceability Links Between Software Documentation and Source Code	Marcus, Maletic, Sergeyev	LSI (N/A)	Univ: LEDA (228-803), Stud: Albergate (73)	Level 1
44	Recovering Code to Documentation Links in OO Systems	Antoniol, Canfora, De Lucia, Marlo	BIM	Univ: LEDA (296)	Level 1
40	Fine grained indexing of software repositories to support impact analysis	Canfora, Cerulo	BM25	OS: Gedit (233), ArgoUML (2208), Firefox (680)	Level 1
38	ADAMS Re-Trace: A Traceability Recovery Tool	De Lucia, Fasano, Oliveto, Tortora	LSI (N/A)	Stud: (48, 50, 54, 55, 73, 74, 111)	Level 4 (7 proj.)
36	On the Equivalence of Information Retrieval Methods for Automated Traceability Link Recovery	Oliveto, Gethers, Poshyvanyk, De Lucia	VSM, LSI (N/A), LM, LDA (t=50-300)	Stud: EasyClinic (77), eTour (174)	Level 1
33	Incremental Approach and User Feedbacks: a Silver Bullet for Traceability Recovery	De Lucia, Oliveto, Sgueglia	VSM, LSI (k=10, **19**, (MODIS), 60 (EasyClinic))	Ind: MODIS (68), Stud: EasyClinic (150)	Level 1
30	A machine learning approach for tracing regulatory codes to product specific requirements	Cleland-Huang, Czauderna, Gibiec, Emenecker	PIN	Mixed: (254)	Level 2
30	Assessing IR-based traceability recovery tools through controlled experiments	De Lucia, Oliveto, Tortora	LSI (N/A)	Stud: EasyClinic (150)	Level 3 (20, 12 subj.)
29	A Traceability Technique for Specifications	Abadi, Nisenson, Simionovici	VSM, LSI (k=5-100, **16** (SCA), **96** (CORBA)), PLSI (k=5-128), SDR (k=5-128), LM	OS: SCA (1311), CORBA (3340)	Level 2
29	Can Information Retrieval Techniques Effectively Support Traceability Link Recovery?	De Lucia, Fasano, Oliveto, Tortora	LSI (k=20%)	Stud: EasyClinic (150), Univ: ADAMS (309), LEDA (803)	Level 1, Level 4 (150 subj.)
29	Software traceability with topic modeling	Asuncion, Asuncion, Taylor	LSI (k=10), LDA (t=10, 20,30)	Univ: ArchStudio (N/A), Stud: EasyClinic (160) Stud: EasyClinic (160)	Level 1
29	Speeding up Requirements to Management in a Product Software Company: Linking Customer Wishes Product Requirements through Linguistic Engineering	Natt och Dag, Gervasi, Brinkkemper, Regnell	VSM	Ind: Baan (12083)	Level 2
29	Tracing Object-Oriented Code into Functional Requirements	Antoniol, Canfora, De Lucia, Casazza, Merlo	BIM	Stud: Albergate (76)	Level 1
28	Clustering support for automated tracing	Duan, Cleland-Huang	PIN	Univ: IBS (185)	Level 1
27	Text mining for software engineering: how analyst feedback impacts final results	Huffman Hayes, Dekhtyar, Sundaram	N/A	Ind: MODIS (68)	Level 3 (3 subj.)
26	A feasibility study of automated natural language requirements analysis in market-driven development	Natt och Dag, Regnell, Carlshamre, Andersson, Karlsson	VSM	Ind: Telelogic (1891, 1089)	Level 1
26	Implementation of an Efficient Requirements Analysis Supporting System Using Similarity Measure Techniques	Park, Kim, Ko, Seo	Sliding window, syntactic parser	Ind: Unclear (33)	Level 1
25	Traceability Recovery in RAD Software Systems	Di Penta, Gradara, Antoniol	BIM	Univ: TM (49)	Level 1
23	REquirements TRacing On target (RETRO): improving software maintenance through traceability recovery	Huffman Hayes, Dekhtyar, Sundaram, Holbrook, Vadlamudi, April	VSM	Ind: CM-1 (74)	Level 3 (30 subj.)
22	Phrasing in Dynamic Requirements Trace Retrieval	Zou, Settimi, Cleland-Huang	PIN	Univ: IBS (235), LC (59), EBT (93)	Level 1

Table 10: Classification of primary publications, part III.

Cit.	Title	Authors	IR mod.	Dataset	Evaluation
21	Combining Textual and Structural Analysis of Software Artifacts for Traceability Link Recovery	McMillan, Poshyvanyk, Revelle	LSI (k=15, 25, 50, **75**)	Univ: CoffeeMaker (143)	Level 1
20	Tracing requirements to defect reports: an application of information retrieval techniques	Yadla, Huffman Hayes, Dekhtyar	VSM	Ind: CM-1 (68,118)	Level 2
18	Automated Requirements Traceability: the Study of Human Analysts	Cuddeback, Dekhtyar, Huffman Hayes	VSM	OS: BlueJ Plugin (49)	Level 3 (26 subj.)
18	Incremental Latent Semantic Indexing for Automatic Traceability Link Evolution Management	Jiang, Nguyen, Chen, Jaygarl, Chang	LSI (k=10%)	Univ: LEDA (634)	Level 1
18	Understanding how the requirements are implemented in source code	Zhao, Zhang, Liu, Juo, Sun	VSM	OS: Desktop Calculator (123)	Level 1
17	Improving Automated Requirements Trace Retrieval: A Study of Term-Based Enhancement Methods	Zou, Settimi, Cleland-Huang	PIN	Ind: CM-1 (455), Univ: IBS (235), EBT (93), LC (89), Stud: SE450 (521)	Level 2
17	IR-Based Traceability Recovery Processes: An Empirical Comparison of "One-Shot" and Incremental Processes	De Lucia, Oliveto, Tortora	LSI (N/A)	Stud: EasyClinic (150)	Level 3 (30 subj.)
17	Make the Most of Your Time: How Should the Analyst Work with Automated Traceability Tools?	Dekhtyar, Huffman Hayes, Larsen	VSM	Ind: CM-1 (455)	Level 2
16	Baselines in requirements tracing	Sundaram, Huffman Hayes, Dekhtyar	VSM, LSI (k=**10**,19,29 (MODIS), 100,200 (CM-1))	Ind: CM-1 (455), MODIS (68)	Level 2
11	Challenges for semi-automatic trace recovery in the automotive domain	Leuser	VSM, LSI (N/A)	Ind: Daimler AG (1500)	Level 1
11	Monitoring Requirements Coverage Using Reconstructed Views: An Industrial Case Study	Lormans, Gross, van Deursen, Stehouwer, van Solingen	LSI (N/A)	Ind: LogicaCMG (219)	Level 1
11	On the role of the nouns in IR-based traceability recovery	Capobianco, De Lucia, Oliveto, Panichella, Panichella	LSI (N/A), LM	Stud: EasyClinic (150)	Level 1
10	An experiment on linguistic tool support for consolidation of requirements from multiple sources in market-driven product development	Natt och Dag, Thelin, Regnell	VSM	Stud: PUSS (299)	Level 3 (23 subj.)
9	An Industrial Case Study in Reconstructing Requirements Views	Lormans, van Deursen, Gross	LSI (k=40%)	Ind: LogicaCMG (293)	Level 1
9	Towards Mining Replacement Queries for Hard-to-Retrieve Traces	Gibiec, Czauderna, Cleland-Huang	VSM	Mixed: (254)	Level 2
8	Recovering Relationships between Documentation and Source Code based on the Charecteristics of Software Engineering	Wang, Lai, Liu	LSI (N/A), BIM	Univ: LEDA (597), Univ: IBS (270)	Level 1
8	Trace retrieval for evolving artifacts	Winkler	LSI (k=15%)	Ind: Robert Bosch GmbH (500), MODIS (68)	Level 1
8	Traceability Recovery using Numerical Analysis	Capobianco, De Lucia, Oliveto, Panichella, Panichella	VSM, LSI (N/A), LM, B-splines	Stud: EasyClinic (150)	Level 1
7	Assessing Traceability of Software Engineering Artifacts	Sundaram, Huffman Hayes, Dekhtyar, Holbrook	VSM, LSI (k=10,25, 30,40,60 (MODIS), 10,25,100,200, 400 (CM-1), 5,10,15,25,40 (Waterloo)	Ind: MODIS (68), CM-1 (455), Stud: 22* Waterloo (65)	Level 2
7	Requirement-centric traceability for change impact analysis: A case study	Li, Li, Yang, Li	VSM	Unclear: Requirements Management System (501)	Level 4 (5 subj.)

Table 11: Classification of primary publications, part IV.

Cit.	Title	Authors	IR mod.	Dataset	Evaluation
6	How do we trace requirements: an initial study of analyst behavior in trace validation tasks	Kong, Huffman Hayes, Dekhtyar, Holden	N/A	OS: BlueJ plugin (49)	Level 3 (13 subj.)
6	Technique Integration for Requirements Assesment	Dekhtyar, Huffman Hayes, Sundaram, Holbrook, Dekhtyar	VSM, LSI (N/A), BIM LDA (N/A), Chi2 key extr.	Ind: CM-1 (455)	Level 1
4	Application of Swarm Techniques for Requirements Engineering: Requirements Tracing	Sultanov, Huffman Hayes	VSM, Swarm	Ind: CM-1 (455), Univ: PINE (182)	Level 1
4	On Integrating Orthogonal Information Retrieval Methods to Improve Traceability Recovery	Gethers, Oliveto, Posyvanyk, De Lucia	VSM, LM, RTM	Stud: eAnsi (194), eAnsi (67), EasyClinic (57) EasyClinic (100), eTour (232), SMOS (167)	Level 1
3	A clustering-based approach for tracing object-oriented design to requirement	Zhou, Yu	VSM	Univ: Resource Management Software (33)	Level 1
3	Evaluating the Use of Project Glossaries in Automated Trace Retrieval	Zou, Settimi, Cleland-Huang	PIN	Ind: CM-1 (455), Univ: IBS (235), Stud: SE450 (61)	Level 1
3	On Human Analyst Performance in Assisted Requirements Tracing: Statistical Analysis	Dekhtyar, Dekhtyar, Holden, Huffman Hayes, Cuddeback, Kong	VSM	OS: BlueJ (49)	Level 3 (84 subj.)
3	Tackling Semi-automatic Trace Recovery for Large Specifications	Leuser, Ott	VSM	Ind: Daimler (2095, 944)	Level 1
2	Extraction and visualization of traceability relationships between documents and source code	Chen	Unclear	OS: JDK1.5 (N/A), uDig 1.1.1 (N/A)	Level 1
2	Source code indexing for automated tracing	Mahmoud, Niu	VSM	Stud: eTour (174), iTrust (264)	Level 1
2	Traceability challenge 2011: using tracelab to evaluate the impact of local versus global idf on trace retrieval	Czauderna, Gibiec, Leach, Li, Shin, Keenan, Cleland-Huang	VSM	Ind: CM-1 (75), WV-CCHIT (1180)	Level 2
2	Trust-Based Requirements Traceability	Ali, Guéhéneuc, Antoniol	VSM	OS: Pooka (388), SIP (1853)	Level 1
1	An Adaptive Approach to Impact Analysis from Change Requests to Source Code	Gethers, Kagdi, Dit, Poshyvanyk	LSI (N/A)	OS: ArgoUML (qualitative analysis)	Level 2
1	Do Better IR Tools Improve the Accuracy of Engineers' Traceability Recovery?	Borg, Pfahl	VSM	Ind: CM-1 (455)	Level 3 (8 subj.)
1	Experiences with text mining large collections of unstructured systems development artifacts at JPL	Port, Nikora, Hihn, Huang	LSI (N/A)	Unclear	Level 3
1	Improving Automated Documentation to Code Traceability by Combining Retrieval Techniques	Chen, Grundy	VSM	OS: JDK (431)	Level 1
1	Improving IR-based Traceability Recovery Using Smoothing Filters	De Lucia, Di Penta, Oliveto, Panichella, Panichella	VSM, LSI (N/A)	Univ: PINE (131), Stud: EasyClinic (150)	Level 1
1	Using semantics-enabled information retrieval in requirements tracing: An ongoing experimental investigation	Mahmoud, Niu	VSM	Ind: CM-1 (455)	Level 1
1	Traceclipse: an eclipse plug-in for traceability link recovery and management	Klock, Gethers, Dit, Poshyvanyk	Unclear	Ind: CM-1 (455), Stud: EasyClinic (150)	Level 1
0	A combination approach for enhancing automated traceability: (NIER track)	Chen, Hosking, Grundy	VSM	OS: JDK 1.5 (N/A)	Level 1

Table 12: Classification of primary publications, part V.

Cit.	Title	Authors	IR mod.	Dataset	Evaluation
0	A Comparative Study of Document Correlation Techniques for Traceability Analysis	Parvathy, Vasudevan, Balakrishnan	VSM, LSI (k=10), LDA (t=21), CTM	Unclear: (43), (261)	Level 1
0	A requirement traceability refinement method based on relevance feedback	Kong, Li, Li, Yang, Wang	VSM, LM	Ind: Web app (511)	Level 1
0	An Improving Approach for Recovering Requirements-to-Design Traceability Links	Di, Zhang	BIM	Ind: CM-1 (455), MODIS (68)	Level 1
0	Proximity-based traceability: An empirical validation using ranked retrieval and set-based measures	Kong, Huffman Hayes	VSM	Ind: CM-1 (75), OS: Pine (182), Univ: StyleChecker (49), Stud: EasyClinic (77)	Level 2
0	Reconstructing Traceability between Bugs and Test Cases: An Experimental Study	Kaushik, Tahvildari, Moore	LSI (k=50-500, **150-200**)	Ind: RIM (13389)	Level 1
0	Requirements Traceability for Object Oriented Systems by Partitioning Source Code	Ali, Guéhéneuc, Antoniol	VSM	OS: Pooka (388), SIP (1853), Univ: iTrust (526)	Level 1
0	Software verification and validation research laboratory (SVVRL) of the University of Kentucky: traceability challenge 2011: language translation	Huffman Hayes, Sultanov, Kong, Li	VSM	Stud: EasyClinic (150), eTour (174)	Level 2
0	The role of the coverage analysis during IR-based traceability recovery: A controlled experiment	De Lucia, Oliveto, Tortora	LSI (N/A)	Stud: EasyClinic (150)	Level 3 (30 subj.)
0	Towards a Benchmark for Traceability	Ben Charrada, Casper, Jeanneret, Glinz	VSM	Univ: AquaLush (793)	Level 1

Table 13: Most productive authors and affiliations. For authors, the first number is the total number of primary publications, while the number in parenthesis is first-authored primary publications. For affiliations, the numbers show the number of primary publications first-authored by an affiliated researcher.

Author	**Publications**
Andrea De Lucia	16 (9)
Jane Huffman Hayes	16 (6)
Alexander Dekhtyar	15 (3)
Rocco Oliveto	13 (1)
Jane Cleland-Huang	10 (3)
Affiliation	**Publications**
University of Kentucky, United States	13
University of Salerno, Italy	11
DePaul University, United States	10
University of Sannio, Italy	5

PART II: THE SOLUTION PHASE

CHAPTER III

AUTOMATED BUG ASSIGNMENT: ENSEMBLE-BASED MACHINE LEARNING IN LARGE SCALE INDUSTRIAL CONTEXTS

Abstract

Context: Bug report assignment is an important part of software maintenance. In particular, incorrect assignments of bug reports to development teams can be very expensive in large software development projects. Several studies propose automating bug assignment techniques using machine learning in open source software contexts, but no study exists for large-scale proprietary projects in industry. **Objective:** The goal of this study is to evaluate automated bug assignment techniques that are based on machine learning classification. In particular, we study the state-of-the-art ensemble learner Stacked Generalization (SG) that combines several classifiers. **Method:** We collect more than 50,000 bug reports from five development projects from two companies in different domains. We implement automated bug assignment and evaluate the performance in a set of controlled experiments. **Results:** We show that SG scales to large scale industrial application and that it outperforms the use of individual classifiers for bug assignment, reaching prediction accuracies from 50% to 90% when large training sets are used. In addition, we show how old training data can decrease the prediction accuracy of bug assignment. **Conclusions:** We advice industry to use SG for bug assignment in proprietary contexts, using at least 2,000 bug reports for training. Finally, we highlight the importance of not solely relying on results from cross-validation when evaluating automated bug assignment.

Leif Jonsson, Markus Borg, David Broman, Kristian Sandahl, Sigrid Eldh, and Per Runeson *Under Revision in Empirical Software Engineering*

1 Introduction

In large projects, the continuous inflow of bug reports[1] challenges the developers' abilities to overview the content of the Bug Tracking System (BTS) [53, 257]. As a first step toward correcting a bug, the corresponding bug report must be assigned to a development team or an individual developer. This task, referred to as *bug assignment*, is normally done manually. However, several studies report that manual bug assignment is labor-intensive and error-prone [44, 55, 248], resulting in "bug tossing" (i.e., reassigning bug reports to another developer) and delayed bug corrections. Previous work report that bug tossing is frequent in large projects; 25% of bug reports are reassigned in the Eclipse Platform project [24] and over 90% of the fixed bugs in both the Eclipse Platform project and in projects in the Mozilla foundation have been reassigned at least once [55]. Moreover, we have previously highlighted the same phenomenon in large-scale maintenance at Ericsson [255].

Several researchers have proposed improving the situation by automating bug assignment. The most common automation approach is based on *classification* using supervised Machine Learning (ML) [8, 23, 248] (see Section 2 for a discussion about machine learning and classification). By training a classifier, incoming bug reports can automatically be assigned to developers. A wide variety of classifiers have been suggested, and previous studies report promising prediction accuracies ranging from 40% to 60% [6, 23, 248, 308]. Previous work has focused on Open Source Software (OSS) development projects, especially the Eclipse and Mozilla projects. Only a few studies on bug assignment in proprietary development projects are available, and they target small organizations [217, 308]. Although OSS development is a relevant context to study, it differs from proprietary development in aspects such as development processes, team structure, and developer incentives. Consequently, whether previous research on automated bug assignment applies to large proprietary development organizations remains an open question.

Researchers have evaluated several different ML techniques for classifying bug reports. The two most popular classifiers in bug assignment are Naïve Bayes (NB) and Support Vector Machines (SVM), applied in pioneering work by Čubranić *et al.* [125] and Anvik *et al.* [23], respectively. Previous work on bug assignment has also evaluated several other classifiers, and compared the *prediction accuracy* (i.e., the proportion of bug reports assigned to the correct developer) with varying results [6, 23, 24, 55, 217]. To improve the accuracy, some authors have presented customized solutions for bug assignment, tailored for their specific project con-

[1] Other common names for bug report include *issues*, *tickets*, *fault reports*, *trouble reports*, *defect reports*, *anomaly reports*, *maintenance requests*, and *incidents*.

texts (e.g., Xie *et al.* [484] and Xia *et al.* [482]). While such approaches have the potential to outperform general purpose classifiers, we instead focus on a solution that can be deployed as a plug-in to an industrial BTS with limited customization. On the other hand, our solution still provides a novel technical contribution in relation to previous work on ML-based bug assignment by *combining* individual classifiers.

Studies in other domains report that *ensemble learners*, an approach to combine classifiers, can outperform individual techniques when there is diversity among the individual classifiers [287]. In recent years, combining classifiers has been used also for applications in software engineering. Examples include effort estimation [303], fault localization [446], and fault classification [482]. In this article, we propose using *Stacked Generalization* (SG) [478] as the ensemble learner for improving prediction accuracy in automated bug assignment. SG is a state-of-the-art method to combine output from multiple classifiers, used in a wide variety of applications. One prominent example was developed by the winning team of the Netflix Prize, where a solution involving SG outperformed the competition in predicting movie ratings, and won the $1 million prize [432]. In the field of software engineering, applications of SG include predicting the numbers of remaining defects in black-box testing [302], and malware detection in smartphones [13]. In a previous pilot study, we initially evaluated using SG for bug assignment with promising results [255]. Building on our previous work, this paper constitutes a deeper study using bug reports from different proprietary contexts. We analyze how the prediction accuracy depends on the choice of individual classifiers used in SG. Furthermore, we study *learning curves* for different systems, that is, how the amount of training data impacts the overall prediction accuracy.

We evaluate our approach of automated bug assignment on bug reports from five large proprietary development projects. Four of the datasets originate from product development projects at a telecom company, totaling more than 35,000 bug reports. To strengthen the external validity of our results, we also study a dataset of 15,000 bug reports, collected from a company developing industrial automation systems. Both development contexts constitute large-scale proprietary software development, involving hundreds of engineers, working with complex embedded systems. As such, we focus on software engineering much different from the OSS application development that has been the target of most previous work. Moreover, while previous work address *bug assignment to individual developers*, we instead evaluate *bug assignment to different development teams*, as our industrial partners report this task to be more important. In large scale industrial development it makes sense to assign bugs to a team and let the developers involved distribute the work internally. Individual developers might be unavailable for a number of reasons, e.g., temporary peaks of workload, sickness, or employee turnover, thus team assignment is regarded as more important by our industry partners.

The overall goal of our research is to *support bug assignment in large proprietary development projects using state-of-the-art ML*. We further refine this goal into four Research Questions (RQ):

RQ1 Does stacked generalization outperform individual classifiers?

RQ2 How does the ensemble selection in SG affect the prediction accuracy?

RQ3 How consistent learning curves does SG display across projects?

RQ4 How does the time locality of training data affect the prediction accuracy?

To be more specific, our contributions are as follows:

- We synthesize results from previous studies on automated bug assignment and present a comprehensive overview (Section 3).
- We present the first empirical studies of automated bug assignment with data originating from large proprietary development contexts, where bug assignments are made at team level (Section 4).
- We conduct a series of experiments to answer the above specified research questions (Section 5) and report the experimental results and analysis from a practical bug assignment perspective (Section 6), including analyzing threats to validity (Section 7).
- We discuss the big picture, that is, the potential to deploy automated support for bug assignment in the two case companies under study (Section 8).

2 Machine Learning

Machine learning is a field of study where computer programs can learn and get better at performing specific tasks by training on historical data. In this section, we discuss more specifically what machine learning means in our context, focusing on *supervised* machine learning – the type of machine learning technique used in this paper.

2.1 Supervised Machine Learning Techniques and Their Evaluation

In *supervised learning*, a machine learning algorithm is *trained* on a *training set* [62]. A training set is a subset of some historical data that is collected over time. Another subset of the historical data is the *test set*, used for *evaluation*. The evaluation determines how well the system performs with respect to some metric.

In our context, an example metric is the number of bug reports that are assigned to correct development teams, that is, the teams that ended up solving the bugs. The training set can, in turn, be split into disjoint sets for parameter optimization. These sets are called *hold-out* or *validation* sets. After the system has been trained on the training data, the system is then evaluated on each of the instances in the test set. From the point of view of the system, the test instances are completely new since none of the instances in the training set are part of the test set.

To evaluate the predictions, we apply cross-validation with stratification [281]. Stratification means that the instances in the training sets and the test sets are selected to be proportional to their distribution in the whole dataset. In our experiments, we use stratified *10-fold cross-validation*, where the dataset is split into ten stratified sets. Training and evaluation are then performed ten times, each time shifting the set used for testing. The final estimate of accuracy of the system is the average of these ten evaluations.

In addition to 10-fold cross-validation, we use two versions of timed evaluation to closely replicate a real world scenario: sliding window and cumulative time window. In the *sliding window* evaluation, both the training set and the test set have fixed sizes, but the time difference between the sets varies by selecting the training set further back in time. Sliding window is described in more details in Section 5.5. The sliding window approach makes it possible to study how time locality of bug reports affects the prediction accuracy of a system.

The *cumulative time window* evaluation also has a fixed sized test set, but increases the size of the training set by adding more data further back in time. This scheme is described in more details in Section 5.5. By adding more bug reports incrementally, we can study if adding older bug reports is detrimental to prediction accuracy.

2.2 Classification

We are mainly concerned with the type of machine learning techniques called *classification* techniques. In classification, a software component, called a *classifier*, is invoked with inputs that are named *features*. Features are extracted from the training data instances. Features can, for instance, be in the form of free text, numbers, or nominal values. As an example, an instance of a bug report can be represented in terms of features where the subject and description are free texts, the customer is a nominal value from a list of possible customers, and the severity of the bug is represented on an ordinal scale. In the evaluation phase, the classifier will – based on the values of the features of a particular instance – return the class that the features correspond to. In our case, the different classes correspond to the development teams in the organization that we want to assign bugs to. The features can vary from organization to organization, depending on which data that is collected in the bug tracking system.

2.3 Ensemble Techniques and Stacked Generalization

It is often beneficial to combine the results of several individual classifiers. The general idea to combine classifiers is called *ensemble techniques*. Classifiers can be combined in several different ways. In one ensemble technique, called *bagging* [85], many instances of *the same type* of classifier are trained on *different versions of the training set*. Each classifier is trained on a new dataset, created by sampling with replacement from the original dataset. The final result is then obtained by averaging the results from all of the classifiers in the ensemble. Another ensemble technique, called *boosting*, also involves training several instances of the same type of classifier on a modified training set, which places *different weights* on the different training instances. The classifiers are trained and evaluated in sequence with subsequent classifiers trained with higher weights on instances that previous classifiers have misclassified. A popular version of boosting is called Adaboost [190]. Both bagging and boosting use the same type of classifiers in the ensemble and vary the data the classifiers are trained on.

Stacked Generalization (SG) [478] (also called *stacking* or *blending*) is an ensemble technique that combines several level-0 classifiers *of different types* with one level-1 classifier (see Fig. 1) into an ensemble. The level-1 classifier trains and evaluates all of the level-0 classifiers on *the same data* and learns (using a separate learning algorithm) which of the underlying classifiers (the level-0 classifiers) that perform well on different classes and data. The level-1 training algorithm is typically a relatively simple smooth linear model [474], such as logistic regression. Note that in stacking, it is completely permissible to put other ensemble techniques as level-0 classifiers.

In this study (see Sections 5 and 6), we are using stacked generalization because this ensemble technique meets our goal of combining and evaluating *different* classifiers.

3 Related Work on Automated Bug Assignment

Several researchers have proposed automated support for bug assignment. Most previous work can either be classified as ML classification problems or *Information Retrieval* (IR) problems. ML-based bug assignment uses supervised learning to classify bug reports to the most relevant developer. IR-based bug assignment on the other hand, considers bug reports as queries and developers as documents of various relevance given the query. A handful of recent studies show that specialized solutions for automated bug assignment can outperform both ML and IR approaches, e.g., by combining information in the BTS with the source code repository, or by crafting tailored algorithms for matching bug reports and developers. We focus the review of previous work on applications of off-the-shelf classification algorithms, as our aim is to explore combinations of readily available classifiers.

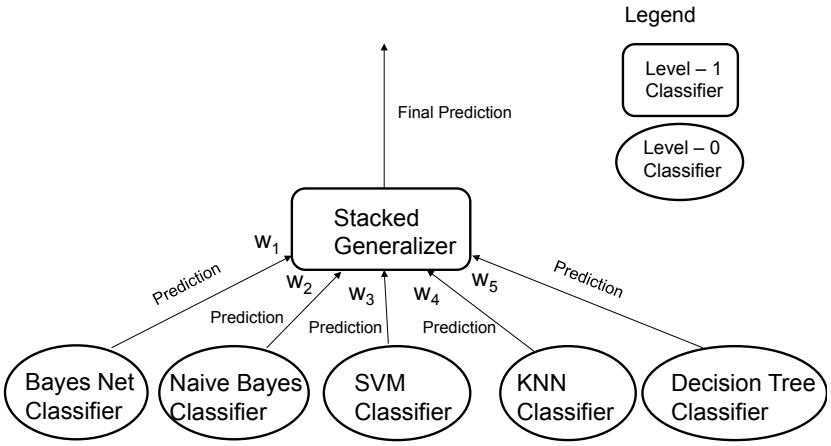


Figure 1: Stacked Generalization

However, we also report key publications both from IR-based bug assignment and specialized state-of-the-art tools for bug assignment in Section 3.2.

3.1 Automated Bug Assignment Using General Purpose Classifiers

Previous work on ML-based bug assignment has evaluated several techniques. Table 1 gives a comprehensive summary of the classifiers used in previous work on ML-based bug assignment. Čubranić *et al.* [125] pioneered the work by proposing a Naïve Bayes (NB) classifier trained for the task. Anvik *et al.* [23] also used NB, but also introduced Support Vector Machines (SVM), and C4.5 classifiers. Later, they extended that work and evaluated also rules-based classification and Expectation Maximization (EM) [21], as well as Nearest Neighbor (NN) [24]. Several other researchers continued the work by Anvik *et al.* by evaluating classification-based bug assignment on bug reports from different projects, using a variety of classifiers. Ahsan *et al.* [6] were the first to introduce Decision Trees (DT), RBF Network (RBF), REPTree (RT), and Random Forest (RF) for bug assignment. The same year, Jeong *et al.* [248] proposed to use Bayesian Networks (BNet). Helming *et al.* [217] used Neural Networks (NNet) and Constant Classifier (CC). In our work, we evaluate 28 different classifiers, as presented in Section 5.

Two general purpose classification techniques have been used more than the others, namely NB and SVM (cf. Table 1). The numerous studies on NB and SVM are in line with ML work in general; NB and SVM are two standard classifiers with often good results that can be considered default choices when exploring a new task. Other classifiers used in at least three previous studies on bug assignment

Table 1: Techniques used in previous studies on ML-based bug assignment. Bold author names indicate comparative studies, capital **X** shows the classifier giving the best results. IR indicates Information Retrieval techniques. The last row shows the study presented in this paper.

	ASSIGNMENT TECHNIQUE																FEATURES									
	Classification													IR			Textual				Nominal					
	Decision Trees	Naive Bayes	Bayesian NETworks	Nearest Neighbor	Neural Networks	RBF Network	Random Forest	REPTree	SVM	Rules	Expect. Maximization	C4.5	Constant Classifier	Algebreic	Probabilistic	RSSE	Title	Description	Comments	Identifiers	Product	Component	Platform	Version	Type	Phase	Priority	Submitter
Ahsan et al. (2009)	x	x				x	x	x	**X**			x					x				x	x	x				
Alenezi et al. (2013)		x															x										
Aljarah et al. (2011)			x														x										
Anvik and Murphy (2011)		x		x					**X**	x	x	x					x	x									
Anvik et al. (2006)		x							**X**			x					x	x									
Baysal. et al. (2009)																x	x	x									
Bhattacharya et al. (2012)		**X**	x						x			x					x	x									
Canfora and Cerulo (2006)															x		x	x	x								
Chen et al. (2011)														x			x	x									
Cubranic and Murphy (2004)		x															x	x									
Helming et al. (2011)	x	x		x	x				**X**				x				x	x									
Jeong et al. (2009)		x	**X**														x	x									
Kagdi et al. (2012)														x					x	x							
Lin et al. (2009)									x			**X**					x	x				x			x	x	x	x
Linares-Vasquez et al. (2012)									x					**X**			x	x	x	x							
Matter et al. (2009)																x		x									
Nagwani and Verma (2012)														x			x	x									
Park et al. (2011)																x		x					x	x		x	
Shokripour et al. (2012)														x					x								
Xuan et al. (2010)		x															x	x									
Jonsson et al. (2015)	x	x	x				x	x	**X**	x							x	x						x	x		x	x

are Bayesian Networks (BNET), and C4.5. We include both NB and SVM in our study, as presented in Section 5.

Eight of the studies using ML-based bug assignment compare different classifiers. The previously largest comparative studies of general purpose classifiers for bug assignment used seven and six classifiers, respectively [6,217]. We go beyond previous work by comparing more classifiers. Moreover, we propose applying ensemble learning for bug assignment, i.e., combining several different classifiers.

Table 1 also displays the features used to represent bug reports in previous work on ML-based bug assignment. Most previous approaches rely solely on textual information, most often the title and description of bug reports. Only two of the previous studies combine textual and nominal features in their solutions. Ahsan *et al.* [6] include information about product, component, and platform, and Lin *et al.* [308] complement textual information with component, type, phase, priority, and submitter. In our study, we complement textual information by submitter site, submitter type, software revision, and bug priority.

Table 2 shows an overview of the previous evaluations of automated bug assignment (including studies presented in Section 3.2). It is evident that previous work has focused on the context of Open Source Software (OSS) development, as 23 out of 25 studies have studied OSS bug reports. This is in line with general research in empirical software engineering, explained by the appealing availability of large amounts of data and the possibility of replications [404]. While there is large variety within the OSS domain, there are some general differences from proprietary bug management that impact our work. First, the bug databases used in OSS development are typically publicly available; anyone can submit bug reports. Second, Paulson *et al.* [371] report that defects are found and fixed faster in OSS projects. Third, while proprietary development often is organized in *teams*, an OSS development community rather consists of individual developers. Also, the management in a company typically makes an effort to maintain stable teams over time despite employee turnover, while the churn behavior of individual developers in OSS projects is well-known [28, 406]. Consequently, due to the different nature of OSS development, it is not clear to what extent previous findings based on OSS data can be generalized to proprietary contexts. Moreover, we are not aware of any major OSS bug dataset that contains team assignments with which we can directly compare our work. This is unfortunate since it would be interesting to use the same set of tools in the two different contexts.

As the organization of developers in proprietary projects tend to be different from OSS communities, the bug assignment task we study differs accordingly. While all previous work (including the two studies on proprietary development contexts by Lin *et al.* [308] and Helming *et al.* [217]) aim at supporting assignment of bug reports to *individual developers*, we instead address the task of bug assignment to *development teams*. Thus, as the number of development teams is much lower than the number of developers in normal projects, direct comparisons of our results to previous work can not be made. As an example, according to Open HUB[2] (Dec 2014), the number of contributors to some of the studied OSS projects in Table 2 are: Linux kernel (13,343), GNOME (5,888), KDE (4,060), Firefox (3,187), NetBeans (893), gcc (534), Eclipse platform (474), Bugzilla (143), OpenOffice (124), Mylyn (92), ArgoUML (87), Maemo (83), UNICASE (83), jEdit (55), and muCommander (9). Moreover, while the number of bugs resolved in our proprietary datasets is somewhat balanced, contributions in OSS communities tend to follow the "onion model" [3], i.e., the commit distribution is skewed, a few core developers contribute much source code, but most developers contribute only occasionally.

Bug reports from the development of Eclipse are used in 14 out of the 21 studies (cf. Table 2). Still, no set of Eclipse bugs has become the de facto benchmark. Instead, different subsets of bug reports have been used in previous work, containing between 6,500 and 300,000 bug reports. Bug reports originating from OSS

[2]Formerly Ohloh.net, an open public library presenting analyses of OSS projects (www.openhub.net).

Table 2: Evaluations performed in previous studies with BTS focus. Bold author names indicate studies evaluating general purpose ML-based bug assignment. Results are listed in the same order as the systems appear in the fourth column. The last row shows the study presented in this paper, even though it is not directly comparable.

	CONTEXT				BEST RESULTS	
	Prop.	OSS	Univ.	Cases (#bug reports)	Accuracy	Other eval.
Ahsan et al. (2009)		X		Mozilla (1,983)	0.44	
Alenezi et al. (2013)		X		Eclipse (7,561, 6,791), Netbeans (11,311), Maemo (3,505)		F@1: 0.34, 0.35, 0.20, 0.48
Aljarah et al. (2011)		X		Eclipse (38,843)		F@1: 0.57
Anvik and Murphy (2011)		X		Eclipse (6,500), Firefox (3,400), gcc (2,600), MyLyn (700), Bugzilla (850)		F@1: 0.22, 0.02, 0.06, 0.46, 0.10
Anvik et al. (2006)		X		Eclipse (8,655), Firefox (9,752), gcc (2,629)	0.58, 0.64	F@1: 0.006
Baysal. et al. (2009)				Not evaluated		
Bhattacharya et al. (2012)		X		Mozilla (550,000), Eclipse (306,296)	0.70, 0.70	
Canfora and Cerulo (2006)		X		Mozilla (12,477), KDE (14,396)	0.32, 0.59	
Chen et al. (2011)		X		Eclipse (115,058), Mozilla (119,852)		Sign. reduced bug tossing
Cubranic and Murphy (2004)		X		Eclipse (15,859)	0.3	
Helming et al. (2011)	X		X	King's Tale (256), UNICASE (1,191), DOLLI (411)	0.40, 0.30, 0.40	
Jeong et al. (2009)		X		Eclipse (211,822), Mozilla (429,903)		Rc@2: 0.58, 0.56
Kagdi et al. (2012)		X		ArgoUML, Eclipse, Koffice		Qualitative
Lin et al. (2009)	X			SoftPM (2,576)	0.78	
Linares-Vasquez et al. (2012)		X		jEdit (200), ArgoUML (100), muCommander (100)		F@1: 0.05, 0.31, 0.60
Matter et al. (2009)		X		Eclipse (130,769)		F@1: 0.3
Nagwani and Verma (2012)		X		Mozilla		Qualitative
Park et al. (2011)		X		Apache (656), Eclipse (47,862), Linux kernel (968), Mozilla (48,424)	0.70, 0.40, 0.30, 0.65	
Servant and Jones (2012)		X		AspectJ (889)	0.35	
Shokripour et al. (2012)		X		Eclipse (35,140), Mozilla (9,917), GNOME (119,176)	0.31, 0.27, 0.28	
Tamrawi et al. (2011)		X		Firefox (188,139), Eclipse (177,637), Apache (43,162), NetBeans (23,522), FreeDesktop (17,084), Gcc (19,430), Jazz (34,228)	0.30, 0.39, 0.40, 0.29, 0.53, 0.46, 0.30	Sign. faster than ML
Wu et al. (2011)				Firefox (5,195)	0.17	
Xia et al. (2012)		X		Eclipse (34,399), Mozilla (26,046), gcc (5,742), OpenOffice (15,448), NetBeans (26,240)		Rc@5: 0.80, 0.56, 0.56, 0.48, 0.71
Xie et al. (2013)		X		Eclipse (2,558), Firefox (3,174)	0.14 (Ecl.)	Rc@2: 0.20 (Ff.)
Xuan et al. (2010)		X		Eclipse (20,000)	0.2	
Jonsson et al. (2015)	X			Telecom (>35,000) + Automation (15,113)	0.71, 0.57, 0.87, 0.79, 0.50	

development in the Mozilla foundation is the second most studied system, containing up to 550,000 bug reports [55]. While we do not study bug repositories containing 100,000s of bug reports, our work involves much larger datasets than the previously largest study in a proprietary context by [308] (2,576 bug reports). Furthermore, we study bug reports from five different development projects in two different companies.

The most common measure to report the success in previous work is *accuracy*[3], reported in 10 out of 21 studies. As listed in Table 2, prediction accuracies ranging from 0.14 to 0.78 have been reported, with an average of 0.42 and standard deviation of 0.17. This suggests that a rule-of-thumb could be that automated bug assignment has the potential to correctly assign almost every second bug to an individual developer.

3.2 Other Approaches to Automated Bug Assignment

Some previous studies consider bug assignment as an IR problem, meaning that the incoming bug is treated as a search query and the assignment options are the possible documents to retrieve. There are two main families of IR models used in software engineering: algebraic models and probabilistic models [76]. For automated bug assignment, four studies used algebraic models [104, 260, 349, 428]. A probabilistic IR model on the other hand, has only been applied by Canfora and Cerulo [93]. Moreover, only [309] evaluated bug assignment using both classification and IR in the same study, and reported that IR displayed the most promising results.

Most studies on IR-based bug assignment report *F-scores* instead of accuracy. In Table 2 we present F-scores for the first candidate developer suggested in previous work (F@1). The F-scores display large variation; about 0.60 for a study on muCommander and one of the studies of Eclipse, and very low values on work on Firefox, gcc, and jEdit. The variation shows that the potential of automated bug assignment is highly data dependent, as the same approach evaluated on different data can display large differences [24, 309]. A subset of IR-based studies reports neither accuracy nor F-score. Chen *et al.* [104] conclude that their automated bug assignment significantly reduces bug tossing as compared to manual work. Finally, Kagdi *et al.* [260] and Nagwani and Verma [349] perform qualitative evaluations of their approaches. Especially the former study reports positive results.

Three studies on automated bug assignment identified in the literature present tools based on content-based and collaborative filtering, i.e., techniques from research on Recommendation Systems [401]. Park *et al.* [368] developed an RS where bug reports are represented by their textual description extended by the nominal features: platform, version, and development phase. Baysal *et al.* [44]

[3] Equivalent to recall when recommending only the most probable developer, aka. the Top-1 recommendation or Rc@1

presented a framework for recommending developers for a given bug report, using a vector space analysis of the history of previous bug resolutions. Matter *et al.* [332] matched bug reports to developers by modelling the natural language in historical commits and comparing them to the textual content of bug reports.

More recently, some researchers have showed that the accuracy of automated bug assignment can be improved by implementing more advanced algorithms, tailored for both the task and the context. Tamrawi *et al.* [444] proposed Bugzie, an automated bug assignment approach they refer to as fuzzy set and cache-based. Two assumptions guide their work: 1) the textual content of bug reports is assumed to relate to a specific *technical aspect* of the software system, and 2) if a developer frequently resolves bugs related to such a technical aspect, (s)he is capable of resolving related bugs in the future. Bugzie models both technical aspects and developers' expertise as bags-of-words and matches them accordingly. Furthermore, to improve the scalability, Bugzie recommends only developers that recently committed bug resolutions, i.e., developers in the cache. Bugzie was evaluated on more than 500,000 bug reports from seven OSS projects, and achieved an prediction accuracies between 30% and 53%.

Wu *et al.* [481] proposed DREX, an approach to bug assignment using k-nearest neighbour search and social network analysis. DREX recommends performs assignment by: 1) finding textually similar bug reports, 2) extracting developers involved in their resolution, and 3) ranking the developers expertise by analyzing their participation in resolving the similar bugs. The participation is based on developers' comments on historical bug reports, both manually written comments and comments automatically generated when source code changes are committed. DREX uses the comments to construct a social network, and approximated participation using a series of network measures. An evaluation on bug reports from the Firefox OSS project shows the social network analysis of DREX outperforms a purely textual approach, with a prediction accuracy of about 15% and recall when considering the Top-10 recommendations (Rc@10, i.e., the bug is included in the 10 first recommendations) of 0.66.

Servant and Jones [424] developed WhoseFault, a tool that both assigns a bug to a developer and presents a possible location of the fault in the source code. WhoseFault is also different from other approaches reported in this section, as it performs its analysis originating from failures from automated testing instead of textual bug reports. To assign appropriate developers to a failure, WhoseFault combines a framework for automated testing, a fault localization technique, and the commit history of individual developers. By finding the likely position of a fault, and identifying the most active developers of that piece of source code, WhoseFault reaches a prediction accuracy of 35% for the 889 test cases studied in the AspectJ OSS project. Moreover, the tool reports the correct developer among the top-3 recommendations for 81.44% of the test cases.

A trending technique to process and analyze natural language text in software engineering is *topic modeling*. Xie *et al.* [484] use topic models for automated

bug assignment in their approach DRETOM. First, the textual content of bug reports is represented using topic models (Latent Dirichlet Allocation (LDA) [65]). Then, based on the bug-topic distribution, DRETOM maps each bug report to a single topic. Finally, developers and bug reports are associated using a probabilistic model, considering the *interest* and *expertise* of a developer given the specific bug report. DRETOM was evaluated on more than 5,000 bug reports from the Eclipse and Firefox OSS projects, and achieved an accuracy of about 15%. However, considering the Top-5 recommendations the recall reaches 80% and 50% for Eclipse and Firefox, respectively.

Xia *et al.* [482] developed DevRec, a highly specialized tool for automated bug assignment, that also successfully implemented topic models. Similar to the bug assignment implemented in DREX, DevRec first performs a k-nearest neighbours search. DevRec however calculates similarity between bug reports using an advanced combination of the *terms* in the bug reports, its *topic* as extracted by LDA, and the *product* and *component* the bug report is related to (referred to as BR-based analysis). Developers are then connected to bug reports based on multi-label learning using ML-KNN. Furthermore, DevRec then also models the affinity between developers and bug reports by calculating their distances (referred to as D-based analysis). Finally, the BR-analysis and the D-based analyses are combined to recommend developers for new bug reports. Xia *et al.* [482] evaluated DevRec on more than 100,000 bug reports from five OSS projects, and they also implemented the approaches proposed in both DREX and Bugzie to enable a comparison. The authors report average Rc@5 and Rc@10 of 62% and 74%, respectively, constituting considerable improvements compared to both DREX and Bugzie.

In contrast to previous work on specialised tools for bug assignment, we present an approach based on general purpose classifiers. Furthermore, our work uses standard features of bug reports, readily available in a typical BTS. As such, we do not rely on advanced operations such as mining developers' social networks, or data integration with the commit history from a separate source code repository. The reasons for our more conservative approach are fivefold:

1. Our study constitutes initial work on applying ML for automated bug assignment in proprietary contexts. We consider it an appropriate strategy to first evaluate general purpose techniques, and then, if the results are promising, move on to further refine our solutions. However, while we advocate general purpose classifiers in this study, the way we combine them into an ensemble is novel in automated bug assignment.

2. The two proprietary contexts under study are different in terms of work processes and tool chains, thus it would not be possible to develop one specialized bug assignment solution that fits both the organizations.

3. As user studies on automated bug assignment are missing, it is unclear to what extent slight tool improvements are of practical significance for an end

user. Thus, before studies evaluate the interplay between users and tools, it is unclear if specialized solutions are worth the additional development effort required. This is in line with discussions on improved tool support for trace recovery [72], and the difference of *correctness* and *utility* of recommendation systems in software engineering [32].

4. Relying on general purpose classifiers supports transfer of research results to industry. Our industrial partners are experts on developing high quality embedded software systems, but they do not have extensive knowledge of ML. Thus, delivering a highly specialized solution would complicate both the hand-over and the future maintenance of the tool. We expect that this observation generalizes to most software intensive organizations.

5. Using general purpose techniques supports future replications in other companies. As such replications could be used to initiate case studies involving end users, a type of studies currently missing, we believe this to be an important advantage of using general purpose classifiers.

4 Case Descriptions

This section describes the two case companies under study, both of which are bigger than the vast majority of OSS projects. In OSS projects a typical power-law behavior is seen with a few projects, such as the Linux kernel, Mozilla etc, having large number of contributors. We present the companies guided by the six context facets proposed by Petersen and Wohlin [374], namely *product*, *processes*, *practices and techniques*, *people*, *organization*, and *market*. Also, we present a simplified model of the bug handling processes used in the companies. Finally, we illustrate where in the process our machine learning system could be deployed to increase the *level of automation*, as defined by Parasuraman *et al.* [367][4].

4.1 Description of Company Automation

Company Automation is a large international company active in the power and automation sector. The case we study consists of a development organization managing hundreds of engineers, with development sites in Sweden, India, Germany, and the US. The development context is safety-critical embedded development in the domain of industrial control systems, governed by IEC 61511[5]. A typical project has a length of 12-18 months and follows an iterative stage-gate project management model. The software is certified to a Safety Integrity Level (SIL) of 2 as defined by IEC 61508 [239], mandating strict processes on the development

[4]Ten levels of automation, ranging from 0, for fully manual work, to 10, when the computer acts autonomously ignoring the human.

[5]Functional safety - Safety instrumented systems for the process industry sector

and maintenance activities. As specified by IEC 61511 [238], all changes to safety classified source code requires a formal impact analysis before any changes are made. Furthermore, the safety standards mandate that both forward and backward traceability should be maintained during software evolution.

The software product under development is a mature system consisting of large amounts of legacy code; parts of the code base are more than 20 years old. As the company has a low staff turnover, many of the developers of the legacy code are still available within the organization. Most of the software is written in C/C++. Considerable testing takes place to ensure a very high code quality. The typical customers of the software product require safe process automation in very large industrial sites.

The bug-tracking system (BTS) in Company Automation has a central role in the change management and the impact analyses. All software changes, both source code changes and changes to documentation, must be connected to an issue report. Issue reports are categorized as one of the following: *error corrections* (i.e., bug reports), *enhancements*, *document modification*, and *internal* (e.g., changes to test code, internal tools, and administrative changes). Moreover, the formal change impact analyses are documented as attachments to individual issue reports in the BTS.

4.2 Description of Company Telecom

Company Telecom is a major telecommunications vendor based in Sweden. We are studying data from four different development organizations within Company Telecom, consisting of several hundreds of engineers distributed over several countries. Staff turnover is very low and many of the developers are senior developers that have been working on the same products for many years.

The development context is embedded systems in the Information and Communications Technology (ICT) domain. Development in the ICT domain is heavily standardized, and adheres to standards such as 3GPP, 3GPP2, ETSI, IEEE, IETF, ITU, and OMA. Company Telecom is ISO 9001 and TL 9000 certified. At the time the study was conducted, the project model was based on an internal waterfall-like model, but has since then changed to an Agile development process.

Various programming languages are used in the four different products. The majority of the code is written in C++ and Java, but other languages, such as hardware description languages, are also used.

Two of the four products are large systems in the ICT domain, one is a middleware platform, and one is a component system. Two of the products are mature with a code base older than 15 years, whereas the other two products are younger, but still older than eight years. All four products are deployed at customer sites world-wide in the ICT market.

Issue management in the design organization is handled in two separate repositories; one for change requests (planned new features or updates) and one for bug reports. In this study we only use data from the latter, the BTS.

Customer support requests to Company Telecom are handled in a two layered approach with an initial customer service organization dealing with initial requests, called Customer Service Requests (CSR). The task of this organization is to screen incoming requests so that only hardware or software errors and no other issue, such as configuration problems, are sent down to the second layer. If the customer support organization believes a CSR to be a fault in the product, they file a bug report based on the CSR in the second layer BTS. In this way, the second layer organization can focus on issues that are likely to be faults in the software. In spite of this approach, some bug reports can be configuration issues or other problems not directly related to faults in the code. In this study, we have only used data from the second layer BTS, but there is nothing in principle that prevents the same approach to be used on the first layer CSR's. The BTS is the central point in the bug handling process and there are several process descriptions for the various employee roles. Tracking of analysis, implementation proposals, testing, and verification are all coordinated through the BTS.

4.3 State-of-Practice Bug Assignment: A Manual Process

The bug handling process of both Company Automation and Telecom are substantially more complex than the standard process described by Bugzilla [345]. The two processes are characterized by the development contexts of the organizations. Company Automation develops safety-critical systems, and the bug handling process must therefore adhere to safety standards as described in Section 4.1. The standards put strict requirements on how software is allowed to be modified, including rigorous change impact analyses with focus on traceability. In Company Telecom on the other hand, the sheer size of both the system under development and the organization itself are reflected on the bug handling process. The resource allocation in Company Telecom is complex and involves advanced routing in a hierarchical organization to a number of development teams.

We generalize the bug handling processes in the two case companies and present an overview model of the currently manual process in Figure 2. In general, three actors can file bug reports: i) the developers of the systems, ii) the internal testing organization, and iii) customers that file bug reports via helpdesk functions. A submitted bug report starts in a *bug triaging stage*. As the next step, the Change Control Board (CCB) assigns the bug report to a development team for investigation. The leader of the receiving team then assigns the bug report to an individual developer. Unfortunately, the bug reports often end up with the wrong developer, thus *bug tossing* (i.e., bug report re-assignment) is common, especially between teams. The BTS stores information about the bug tossing that takes place.

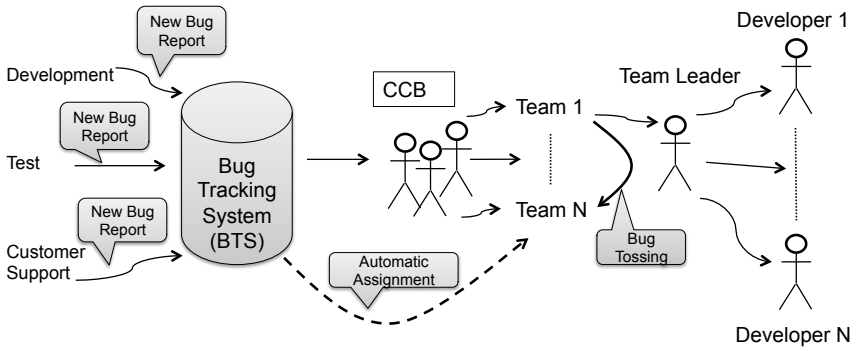


Figure 2: A simplified model of bug assignment in a proprietary context.

As a consequence, we can estimate the classification accuracy of the manual bug assignment process.

4.4 State-of-the-Art: Automated Bug Assignment

We propose, in line with previous work, to automate the bug assignment. Our approach is to use the historical information in the BTS as a labeled training set for a classifier. When a new bug is submitted to the BTS, we encode the available information as features and feed them to a prediction system. The prediction system then classifies the new bug to a specific development team. While this resembles proposals in previous work, our approach differs by: i) aiming at supporting large proprietary organizations, and ii) assigning bug reports to teams rather than individual developers.

Figure 2 shows our automated step as a dashed line. The prediction system offers decision support to the CCB, by suggesting which development team that is the most likely to have the skills required to investigate the issue. This automated support corresponds to a medium level of automation ("the computer suggests one alternative and executes that suggestion if the human approves"), as defined in the established automation model by Parasuraman *et al.* [367].

5 Method

The overall goal of our work is to support bug assignment in large proprietary development projects using state-of-the-art ML. As a step toward this goal, we study five sets of bug reports from two companies (described in Section 4), including information of team assignment for each bug report. We conduct controlled experiments using Weka [211], a mature machine learning environment that is successfully used across several domains, for instance, bioinformatics [186], telecommu-

Table 3: Overview of the research questions, all related to the task of automated team allocation. Each question is listed along with the main purpose of the question, a high-level description of our study approach, and the experimental variables involved.

	RQ1	**RQ2**	**RQ3**	**RQ4**
Description	Does stacked generalization outperform individual classifiers?	How does the ensemble selection in SG affect the prediction accuracy?	How consistent learning curves does SG display across projects?	How does the time locality of training data affect the prediction accuracy?
Rationale	Confirm the result of our previous work [255].	Explore which ensemble selection performs the best.	Study how SG performs on different data, and understand how much training data is required.	Understand how SG should be retrained as new bug reports are submitted.
Approach	Test the hypothesis: "SG does not perform better than individual classifiers wrt. prediction accuracy".	Based on RQ1: evaluate three different ensemble selections.	Using the best ensemble selection from RQ2: evaluate learning curves.	Using the best ensemble selection from RQ2 with amount of training data from RQ3: evaluate SG sensitivity to freshness of training data.
Related experiments	Exp A, Exp B	Exp B	Exp C	Exp D, Exp E
Dependent variable	Prediction accuracy			
Independent variables	Individual classifier	Ensemble selection	Size of training set	Time locality of training data (Exp D), size of training set (Exp E)
Fixed variables	Preprocessing, feature selection, training size		Preprocessing, feature selection, ensemble selection	Preprocessing, feature selection, ensemble selection

nication [11], and astronomy [495]. This section describes the definition, design and setting of the experiments, following the general guidelines by Basili *et al.* [43] and Wohlin *et al.* [477].

5.1 Experiment Definition and Context

The *goal* of the experiments is to study automatic bug assignment using stacked generalization in large proprietary development contexts, for the *purpose* of evaluating its industrial feasibility, from the *perspective* of an applied researcher, planning deployment of the approach in an industrial setting.

Table 3 reminds the reader of our RQs. Also, the table presents the rationale of each RQ, and a high-level description of the research approach we have selected to address them. Moreover, the table maps the RQs to the five sub-experiments we conduct, and the experimental variables involved.

Table 4: Datasets used in the experiments. Note: At the request of our industry partners the table only lists lower bounds for Telecom systems, but the total number sums up to an excess of 50,000 bug reports.

Dataset	#Bug reports	Timespan	#Teams
Automation	15,113	July 2000 – Jan 2012	67
Telecom 1	> 9,000	> 5 years	28
Telecom 2	> 8,000	> 5 years	36
Telecom 3	> 3,000	> 5 years	17
Telecom 4	> 10,000	> 5 years	64
Total	> 50,000		

5.2 Data Collection

We collect data from one development project at Company Automation and four major development projects at Company Telecom. While the bug tracking systems in the two companies show many similarities, some slight variations force us to perform actions to consolidate the input format of the bug reports. For instance, in Company Automation a bug report has a field called "Title", whereas the corresponding field in Company Telecom is called "Heading". We align these variations to make the semantics of the resulting fields the same for all datasets. The total number of bug reports in our study is 15,113 + 35,266 = 50,379. Table 4 shows an overview of the five datasets.

We made an effort to extract similar sets of bug reports from the two companies. However, as the companies use different BTSs, and interact with them according to different processes, slight variations in the extraction steps are inevitable. Company Automation uses a BTS from an external software vendor, while Company Telecom uses an internally developed BTS. Moreover, while the life-cycles of bug reports are similar in the two companies (as described in Section 4.3), they are not equivalent. Another difference is that Company Automation uses the BTS for issue management in a broader sense (incl. new feature development, document updates, and release management), Company Telecom uses the BTS for bug reports exclusively. To harmonize the datasets, we present two separate filtering sequences next, one per company.

Company Automation Data Filtering

The dataset from Company Automation contains in total 26,121 bug reports submitted between July 2000 and January 2012, all related to different versions of the same software system. The bug reports originate from several development projects, and describe issues reported concerning a handful of different related products. During the 12 years of development represented in the dataset, both the

organization and processes have changed towards a more iterative development methodology. We filter the dataset in the following way:

1. We included only CLOSED bug reports to ensure that all bugs have valid team assignments, that is, we filter out bug reports in states such as OPEN, NO ACTION, and CHANGE DEFERRED. This step results in 24,690 remaining bug reports.
2. We exclude bug reports concerning requests for new features, document updates, changes to internal test code, and issues of administrative nature. Thus, we only keep bug reports related to source code of the software system. The rationale for this step is to make the data consistent with Company Telecom, where the BTS solely contains bug reports. The final number of bug reports in the filtered dataset is 15,113.

Company Telecom Data Filtering

Our first step of the data filtering for Company Telecom is to identify a timespan characterized by a stable development process. We select a timespan from the start of the development of the product family to the point in time when an agile development process is introduced [472]. The motivation for this step is to make sure that the study is conducted on a conformed data set. We filter the bug reports in the timespan according to the following steps:

1. We include only bug reports in the state FINISHED.
2. We exclude bug reports marked as duplicates.
3. We exclude bug reports that do not result in an source code update in a product.

After performing these three steps, the data set for the four products contains in total 35,266[6] bug reports.

5.3 ML Framework Selection

To select a platform for our experiments, we study features available in various machine learning toolkits. The focus of the comparison is to find a robust, well tested, and comparatively complete framework. The framework should also include an implementation of stacked generalizer and it should be scalable. As a consequence, we focus on platforms that are suitable for distributed computation. Another criterion is to find a framework that has implemented a large set of state-of-the-art machine learning techniques. With the increased attention of machine learning and data mining, quite a few frameworks have emerged during the last

[6]Due to confidentiality reasons these numbers are not broken down in exact detail per project.

couple of years such as Weka [211], RapidMiner [222], Mahout [364], MOA [57], Mallet [334], Julia [54], and Spark [488] as well as increased visibility of established systems such as SAS, SPSS, MATLAB, and R.

For this study, we select to use a framework called Weka [211]. Weka is a comparatively well documented framework with a public Java API and accompanying book, website, forum, and active community. Weka has many ML algorithms implemented and it is readily extensible. It has several support functionalities, such as cross-validation, stratification, and visualization. Weka has a built-in Java GUI for data exploration and it is also readily available as a stand alone library in JAR format. It has some support for parallelization. Weka supports both batch and online interfaces for some of its algorithms. The meta facilities of the Java language also allows for mechanical extraction of available classifiers. Weka is a well established framework in the research community and its implementation is open source.

5.4 Bug Report Feature Selection

This section describes the feature selection steps that are common to all our data sets. We represent bug reports using a combination of textual and nominal features. Feature selections that are specific to each individual sub-experiment are described together with each experiment.

For the textual features, we limit the number of words because of memory and execution time constraints. To determine a suitable number of words to keep, we run a series of pilot experiments, varying the method and number of words to keep, by varying the built in settings of Weka. We decide to represent the text in the bug reports as the 100 words with highest TF-IDF[7] as calculated by the Weka framework. Furthermore, the textual content of the titles and descriptions are not separated. There are two reasons for our rather simple treatment of the natural language text. First, Weka does not support multiple bags-of-words; such a solution would require significant implementation effort. Second, our focus is not on finding ML configurations that provide the optimal prediction accuracies for our datasets, but rather to explore SG for bug assignment in general. We consider optimization to be an engineering task during deployment.

The non-textual fields available in the two bug tracking systems vary between the companies, leading to some considerations regarding the selection of non-textual features. Bug reports in the BTS of Company Automation contain 79 different fields; about 50% of these fields are either mostly empty or have turned obsolete during the 12 year timespan. Bug reports in the BTS of Company Telecom contain information in more than 100 fields. However, most of these fields are empty when the bug report is submitted. Thus, *we restricted the feature selection to contain only features available at the time of submission of the bug report*,

[7] Term Frequency-Inverse Document Frequency (TF-IDF) is a standard weighting scheme for information retrieval and text mining. This scheme is common in software engineering applications [76].

Table 5: Features used to represent bug reports. For company Telecom the fields are reported for Telecom 1,2,3,4 respectively.

	Company Automation	**Company Telecom**	**Description**
Textual features			
Text	Title+Description	Heading+Observation	One line summary and full description of the bug report
Nominal features			
SubmitterType #Possible values	SubmitterClass 17	Customer >170,>50,>120,>150	Affiliation of the issue submitter
Site #Possible values	SubmitterSite 14	SiteId >250,>60,>80,>200	Site from where the bug was submitted
Revision #Possible values	Revision 103	Faulty revision 547,1325,999,982	Revision of the product that the bug was reported on
Priority #Possible values	Priority 5	Priority 3,3,3,3	Priority of the bug

i.e., features that do not require a deeper analysis effort (e.g., faulty component, function involved). We also want to select a small set of general features, likely to be found in most bug tracking systems. Achieving feasible results using a simple feature selection might simplify industrial adaptation, and also it limits the ML training times. Based on discussions with involved developers, we selected the features presented in Table 5. In the rest of the paper, we follow the nomenclature in the leftmost column.

A recurring discussion when applying ML concerns which features are the best for prediction. In the two bug tracking systems we study, both textual and non-textual features are available, thus we consider it valuable to compare the relative predictive power of the two types of features. While our previous research has indicated that including non-textual features improves the prediction accuracy [255], many other studies rely solely on the text (see Table 1). To motivate the feature selection used in this study, we performed a small study comparing textual vs. non-textual features for our five datasets.

Figure 3 shows the results from our small feature selection experiment. The figure displays results from three experimental runs, all using SG with the best individual classifiers (further described in Section 6.1). The three curves represent three different sets of features: 1) textual and non-textual features, 2) non-textual features only, and 3) textual features only. The results show that for some systems (Telecom 1, 2 and 4) the non-textual features performs better than the textual features alone, while for some systems (Telecom 3 and Automation) the results are the opposite. Thus, our findings strongly suggest that we should combine both

non-textual features and textual features for bug assignment. Note that with more sophisticated text modeling techniques, such as LDA [65], we suspect that the textual features may have a higher impact on the final result.

5.5 Experiment Design and Procedure

Figure 4 shows an overview of our experimental setup. The five datasets originate from two different industrial contexts, as depicted by the two clouds to the left. We implement five sub-experiments (c.f. A-E in Fig. 4), using the Weka machine learning framework. Each sub-experiment is conducted once per dataset, that is, we performed 25 experimental runs. A number of steps implemented in Weka are common for all experimental runs:

1. The complete dataset set of bug reports is imported.
2. Bug reports are divided into training and test sets. In sub-experiments A-C, the bug reports are sampled using stratification.
3. Feature extraction is conducted as specified in Section 5.4.

We executed the experiments on two different computers. We conduct experiments on the Company Automation dataset on a Mac Pro, running Mac OS X 10.7.5, equipped with 24 GB RAM and two Intel(R) Xeon(R) X5670 2.93 GHz CPUs with six cores each. The computer used for the experiments on the Company Telecom datasets had the following specification: Linux 2.6.32.45-0.3-xen, running SUSE LINUX, equipped with eight 2.80 GHz Intel(R) Xeon(R) CPU and 80 GB RAM.

As depicted in Figure 4, there are dependencies among the sub-experiments. Several sub-experiments rely on results from previous experimental runs to select values for both fixed and independent variables. Further details are presented in the descriptions of the individual sub-experiments A–E.

We evaluate the classification using a top-1 approach. That is, we only consider a correct/incorrect classification, i.e., we do not evaluate whether our approach correctly assigns bug reports to a set of candidate teams. In IR evaluations, considering ranked output of search results, it is common to assess the output at different cut-off points, e.g., the top-5 or top-10 search hits. Also some previous studies on bug assignment present top-X evaluations inspired by IR research. However, our reasons for a top-1 approach are three-fold: First, for fully automatic bug assignment a top-1 approach is the only reasonable choice, since an automatic system would not send a bug report to more than one team. Second, a top-1 approach is a conservative choice in the sense that the classification results would only improve with a top-k approach. The third motivation is technical; to ensure high quality evaluations we have chosen to use the built-in mechanisms in the well established Weka. Unfortunately, Weka does not support a top-k approach in its evaluation framework for classifiers.

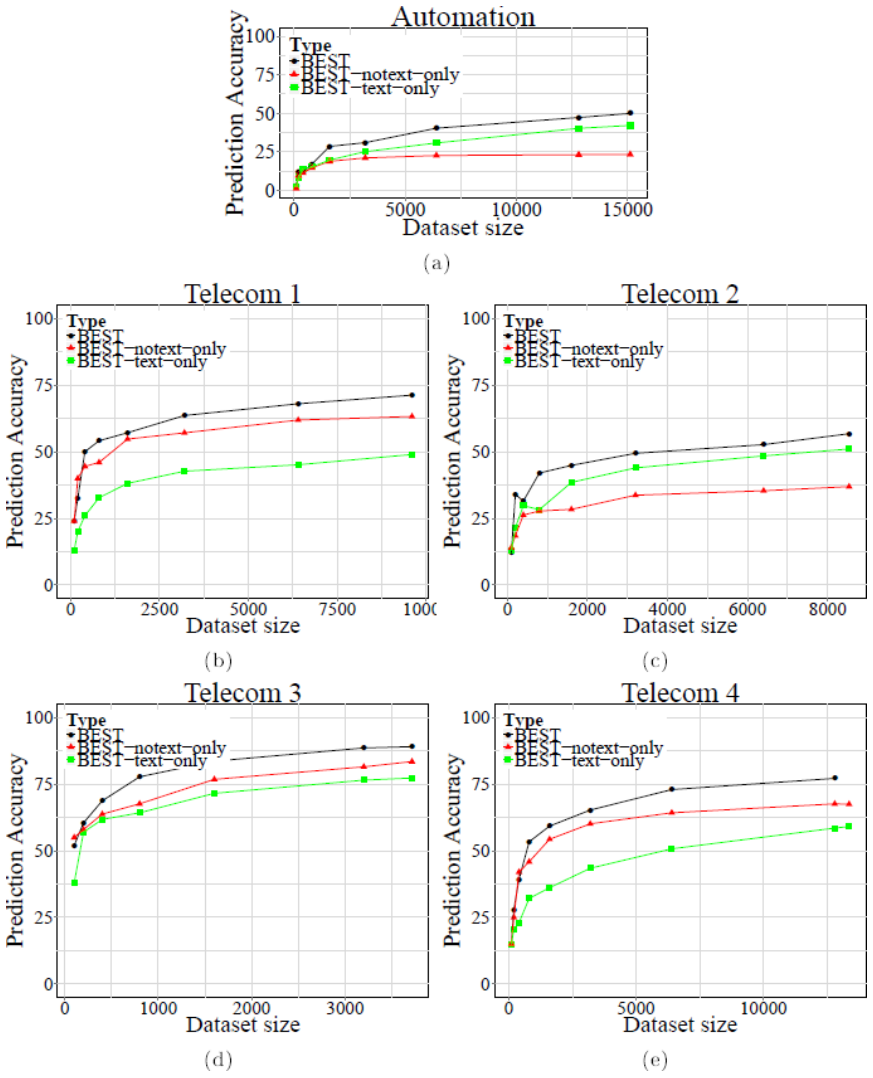


Figure 3: The prediction accuracy when using text only features ("text-only") vs. using non-text features only ("notext-only")

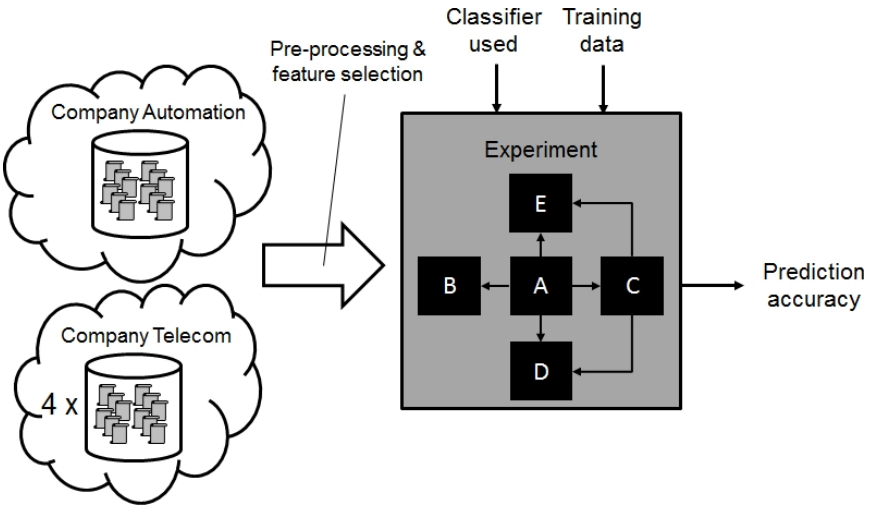


Figure 4: Overview of the controlled experiment. Vertical arrows depict independent variables, whereas the horizontal arrow shows the dependent variable. Arrows within the experiment box depict dependencies between experimental runs A-E: Experiment A determines the composition of individual classifiers in the ensembles studied evaluated in Experiment B-E. The appearance of the learning curves from Experiment C is used to set the size of the time-based evaluations in Experiment D and Experiment E.

Table 6: Individual classifiers available in Weka Development version 3.7.9. Column headings show package names in Weka. Classifiers in bold are excluded from the study because of long training times or exceeding memory constraints.

bayes.	functions.	lazy.	rules.	trees.	misc.
BayesNet	**Logistic**	IBk	DecisionTable	DecisionStump	InputMappedClassifier
NaiveBayes	**MultilayerPerceptron**	KStar	JRip	J48	
NaiveBayesMultinomial	**SimpleLogistic**	**LWL**	OneR	**LMT**	
NaiveBayesMultinomialText	SMO	ZeroR	PART	RandomForest	
NaiveBayesMultinomialUpdateable				RandomTree	
NaiveBayesUpdateable				REPTree	
net.BayesNetGenerator					
net.BIFReader					
net.EditableBayesNet					

Experiment A: Individual Classifiers

Independent variable: Choice of individual classifier

Experiment A investigates RQ1 and the null hypothesis "SG does not perform better than individual classifiers wrt. prediction accuracy". We evaluate the 28 available classifiers in Weka Development version 3.7.9, listed in Table 6. The list of possible classifiers is extracted by first listing all classes in the corresponding .jar file in the "classifier" package and then trying to assign them one by one to a classifier. The independent variable is the *individual classifier*. For all five datasets, we execute 10-fold cross-validation once per classifier. We use all available bug reports in each dataset and evaluated all 28 classifiers on all datasets. The results of this experiment is presented in Section 6.1.

Experiment B: Ensemble Selection

Independent variable: Ensemble selection

Experiment B explores both RQ1 and RQ2, i.e., both if SG is better than individual classifiers and which ensemble of classifiers to choose for bug assignment. As evaluating all combinations of the 28 individual classifiers in Weka is not feasible, we restrict our study to investigate three ensemble selections, each combining five individual classifiers. We chose five as the number of individual classifiers to use in SG at a consensus meeting, based on experiences of prediction accuracy and run-time performance from pilot runs. Moreover, we exclude individual classifiers with run-times longer than 24 hours in Experiment A, e.g., MultiLayerPerceptron and SimpleLogistic.

Based on the results from Experiment A, we select three ensembles for each dataset (cf. Table 6). We refer to these as BEST, WORST, and SELECTED. We chose the first two ensembles to test the hypothesis "combining the best individual classifiers should produce a better result than if you choose the worst". The BEST ensemble consists of the five individual classifiers with the highest prediction ac-

curacy from Experiment A. The WORST ensemble contains the five individual classifiers with the lowest prediction accuracy from Experiment B, while still performing better than the basic classifier *ZeroR* that we see as a lower level baseline. The ZeroR classifier simply always predicts the class with the largest number of bugs. No classifier with a lower classification accuracy than ZeroR is included in any ensemble, thus the ZeroR acts as a lower level cut-off threshold for being included in an ensemble.

The SELECTED ensemble is motivated by a discussion by Wolpert [478], who posits that diversity in the ensemble of classifiers improves prediction results. The general idea is that if you add similar classifiers to a stacked generalizer, less new information is added compared to adding a classifier based on a different classification approach. By having level-0 generalizers of different types, they together will better "span the learning space". This is due to the fundamental theory behind stacked generalization, claiming that the errors of the individual classifiers should average out. Thus, if we use several similar classifiers we do not get the averaging out effect since then, in theory, the classifiers will have the same type of errors and not cancel out. We explore this approach by using the ensemble selection call SELECTED, where we combine the best individual classifiers from five different classification approaches (the packages in Table 6). The results of this experiment is presented in Section 6.1.

Some individual classifiers are never part of a SG. This depends on either that the classifier did not pass the cut-off threshold of being better than the *ZeroR* classifier, this case occurs for instance for the *InputMappedClassifier* (see Table 6). Alternatively the classifier was neither bad enough to be in the WORST ensemble nor good enough to be in the BEST or SELECTED, this is the case with for instance *JRip*.

In all of the ensembles we use SimpleLogistic regression as the level-1 classifier following the general advice by Wolpert [478] and Witten *et al.* [474] of using a relatively simple smooth linear model.

We choose to evaluate the individual classifiers on the whole dataset in favor of evaluating them on a hold-out set, i.e., a set of bug reports that would later not be used in the evaluation of the SG. This is done to maximize the amount of data in the evaluation of the SG. It is important to note that this reuse of data only applies to the selection of which individual classifiers to include in the ensemble. In the evaluation of the SG, all of the individual classifiers are completely retrained on only the training set, and none of the data points in the test set is part of the training set of the individual classifiers. This is also the approach we would suggest for industry adoption, i.e., first evaluate the individual classifiers on the current bug database, and then use them in a SG.

Experiment C: Learning Curves

Independent variable: Amount of training data

The goal of Experiment C is to study RQ3: *How consistent learning curves does SG display across projects?* For each dataset, we evaluate the three ensembles from Experiment B using fixed size subsets of the five datasets: 100, 200, 400, 800, 1600, 3200, 6400, 12800, and ALL bug reports. All subsets are selected using random stratified sampling from the full dataset. As the datasets Telecom 1-3 contain fewer bug reports than 12800, the learning curves are limited accordingly. The results of this experiment is presented in Section 6.2.

Experiment D: Sliding Time Window

Independent variable: Time locality of training data

Experiment D examines RQ4, which addresses how the time locality of the training set affects the prediction accuracy on a given test set. Figure 5 shows an overview of the setup of Experiment D. The idea is to use training sets of the same size increasingly further back in time to predict a given test set. By splitting the chronologically ordered full data set into fixed size training and test sets according to Figure 5, we can generate a new dataset consisting of pairs (x,y). In this dataset, x represents the time difference measured in number of days (delta time) between the start of the training set and the start of the test set. The y variable represents the prediction accuracy of using the training set x days back in time to predict the bug assignments in the selected test set. We can then run a linear regression on the data set of delta time and prediction accuracy samples and examine if there is a negative correlation between delta time and prediction accuracy.

We break down RQ4 further into the following research hypothesis formulation: "Is training data further back in time worse at predicting bug report assignment than training data closer in time"? We test this research hypothesis with the statistical method of simple linear regression. Translated into a statistical hypothesis RQ4 is formulated as:

"Let the difference in time between the submission date of the first bug report in a test set and the submission date of the first bug report in the training set be the independent variable x. Further, let the prediction accuracy on the test set be the dependent variable y. Is the coefficient of the slope of a linear regression fit on x and y statistically different from 0 and negative at the 5 % α level?"

To create the training set and test sets, we sort the complete dataset in chronological order on the bug report submission date. To select suitable sizes to split the training set and test sets, we employ the following procedure. For the simple linear regression, we want to create enough sample points to be able to run a linear regression with enough power to detect a significant difference and still have as large training and test sets as possible to reduce the variance in the generated

samples. Green suggests the following formula [206]: $N \geq 50 + 8m$ as a rule-of-thumb for calculating the needed number of samples at α level of 5 % and β level of 20 %, where m is the number of independent variables. In our case we have one independent variable (delta time) so the minimum number of samples in our case is 58 = 50 + 8 * 1. We use a combination of theoretical calculations for the lower and upper bounds on the number of training samples given that we want an 80/20 ratio of training to test data. We combine the theoretical approach with a program that calculates the number of sample points generated by a given training and test set size, by simulating runs. This combination together with Green's formula let us explore the most suitable training and test sets for the different systems.

We also know from Experiment C that the "elbow" where the prediction accuracy tends to level out is roughly around 1,000-2,000 samples, this together with the calculations for the linear regression guided our decision for the final selection of sample size.

We arrived at the following dataset sizes by exploring various combinations with the simulation program, the theoretical calculations and the experience from Experiment C. For the smallest of the systems, the maximum sizes of training and test sets that gives more than 58 samples amounts to 619 and 154 bug reports respectively. For the larger systems, we can afford to have larger data sets. For comparison we prioritize to have the same sized sets for all the other systems. When we calculate the set sizes for the smallest of the larger systems, we arrived at 1,400 and 350 bug reports for the training and test set sizes, respectively. These values are then chosen for all the other four systems. The results of this analysis is presented in Section 6.3.

Experiment E: Cumulative Time Window

Independent variable: Amount of training data

Experiment E is also designed to investigate RQ4, i.e., how the time locality of the training set affects the prediction accuracy. Instead of varying the training data using a fixed size sliding window as in Experiment D, we fix the starting point and vary the amount of the training data. The independent variable is the cumulatively increasing *amount of training data*. This experimental setup mimics realistic use of SG for automated bug assignment.

Figure 6 depicts an overview of Experiment E. We sort the dataset in chronological order on the issue submission date. Based on the outcome from Experiment C, we split the dataset into a corresponding number of equally sized chunks. We used each chunk as a test set, and for each test set we vary the number of previous chunks used as training set. Thus, the *amount of training data* was the independent variable. We refer to this evaluation approach as *cumulative time window*. Our setup is similar to the "incremental learning" that Bhattacharya *et al.* [55] present in their work on bug assignment, but we conduct a more thorough evalua-

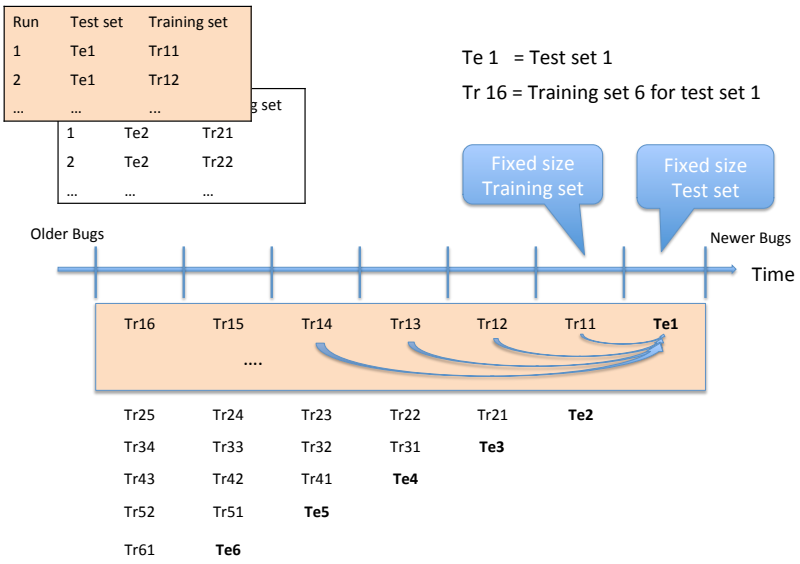


Figure 5: Overview of the time-sorted evaluation. Vertical bars show how we split the chronologically ordered data set into training and test sets. This approach gives us many measurement points in time per test set size. Observe that the *time* between the different sets can vary due to non-uniform bug report inflow but the *number* of bug reports between each vertical bar is fixed.

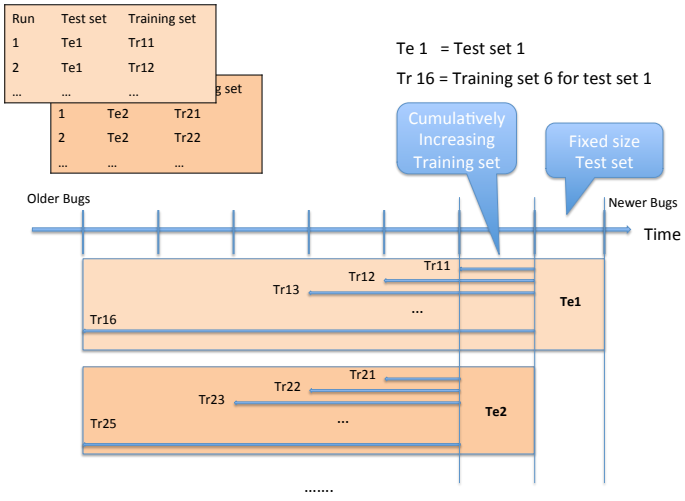


Figure 6: Overview of the cumulative time-sorted evaluation. We use a fixed test set, but cumulatively increase the training set for each run.

tion. We split the data into training and test sets in a more comprehensive manner, and thus conduct several more experimental runs. The results of this experiment is presented in Section 6.4.

6 Results and Analysis

6.1 Experiment A: Individual Classifiers and Experiment B: Ensemble Selection

Experiment A investigates whether SG outperforms individual classifiers. Table 7 shows the individual classifier performance for the five evaluated systems. It also summarizes the results of running SG with the three different configurations BEST, WORST, and SELECTED, related to Experiment B. In Table 7 we can view the classifier "rules.ZeroR" as a sort of lower baseline reference. The ZeroR classifier simply always predicts the class with the highest number of bug reports.

The answer to RQ1 is that *while the improvements in some projects are marginal, using reasonable ensemble selection leads to a better prediction accuracy than using any of the individual classifiers*. On our systems, the improvement is 3% better than the best of the individual classifiers on two of the systems. The best improve-

Table 7: Individual classifier results (rounded to two digits) on the five systems use the full data set and 10-fold cross validation. Out of memory is marked O-MEM and an execution that exceeds a time threshold is marked O-TIME.

Classifier	Accuracy Automation	Accuracy Telecom 1	Accuracy Telecom 2	Accuracy Telecom 3	Accuracy Telecom 4
bayes.BayesNet	35 % (B,S)	O-MEM	O-MEM	O-MEM	O-MEM
bayes.NaiveBayes	15 % (W)	25 % (W)	18 % (W)	35 %	17 %
bayes.NaiveBayesMultinomial	22 %	34 % (W)	32 %	53 % (W)	26 % (W)
bayes.NaiveBayesMultinomialText	6 %	13 %	16 %	43 %	19 %
bayes.NaiveBayesMultinomialUpdateable	26 %	34 %	32 % (S)	61 % (W)	28 % (W)
bayes.NaiveBayesUpdateable	15 %	25 %	18 %	35 %	17 %
bayes.net.BIFReader	35 %	O-MEM	O-MEM	O-MEM	O-MEM
bayes.net.BayesNetGenerator	35 %	41 % (S)	31 % (W)	66 % (S,W)	37 % (S)
bayes.net.EditableBayesNet	35 %	O-MEM	O-MEM	O-MEM	O-MEM
functions.SMO	**42 %** (B,S)	**70 %** (B,S)	**54 %** (B,S)	**86 %** (B,S)	**78 %** (B,S)
lazy.IBk	38 % (B)	58 % (S)	44 % (B)	77 % (B)	63 %
lazy.KStar	42 % (B,S)	50 %	46 % (B,S)	77 % (S)	60 % (S)
lazy.LWL	9 % (W)	21 % (W)	O-MEM	O-MEM	O-MEM
misc.InputMappedClassifier	6 %	13 %	16 %	43 %	19 %
rules.DecisionTable	26 %	52 %	31 % (W)	65 % (W)	55 %
rules.JRip	23 %	51 %	36 %	73 %	55 %
rules.OneR	13 % (W)	43 % (W)	30 % (W)	71 %	50 % (W)
rules.PART	29 % (S)	61 % (B,S)	38 % (S)	76 % (S)	64 % (B,S)
rules.ZeroR	6 %	13 %	16 %	43 %	19 %
trees.DecisionStump	7 % (W)	21 % (W)	22 % (W)	44 % (W)	20 % (W)
trees.J48	30 %	62 % (B)	40 % (B)	78 % (B)	66 % (B)
trees.LMT	O-MEM	O-MEM	O-MEM	O-MEM	O-MEM
trees.REPTree	29 %	62 % (B)	34 %	79 % (B)	67 % (B)
trees.RandomForest	39 % (B,S)	63 % (B,S)	49 % (B,S)	84 (B,S)%	67 % (S)
trees.RandomTree	27 %	52 %	32 %	69 %	49 % (W)
functions.Logistic	O-MEM	O-MEM	O-MEM	O-MEM	O-MEM
functions.SimpleLogistic	40 %	O-TIME	52 %	O-TIME	O-TIME
functions.MultilayerPerceptron	20 % (W)	O-TIME	O-TIME	O-TIME	O-TIME
SG BEST (B)	50 %	71 %	57 %	89 %	77 %
SG SELECTED (S)	50 %	71 %	57 %	89 %	79 %
SG WORST (W)	28 %	57 %	45 %	83 %	62 %

ment is 8% on the Automation system and the smallest improvement is 1% on system Telecom 1 and 4, which can be considered negligible. This conclusion must be followed by a slight warning; mindless ensemble selection together with bad luck can lead to *worse* result than some of the individual classifiers. In none of our runs (including with the WORST ensemble) is the stacked generalizer worse than all of the individual classifiers.

Experiment B addresses different ensemble selections in SG. From Table 7 we see that in the cases of the BEST and SELECTED configurations the stacked generalizer in general performs as well, or better, than the individual classifiers. In the case of Telecom 1 and 4, there is a negligible difference between the best individual classifier SMO and the SELECTED and BEST SG. We also see that when we use the WORST configuration the result of the stacked generalizer is worse than the best of the individual classifiers, but it still performs better than some of the individual classifiers. When it comes to the individual classifiers we

note that the SMO classifier performs best on all systems. The conclusion is that the SG does not do worse than any of the individual classifiers but can sometimes perform better.

Figure 7 shows the learning curves (further presented in relation to Experiment C) for the five datasets using the three configurations BEST, WORST, and SELECTED. The figures illustrate that the two ensembles BEST and SELECTED have very similar performance across the five systems. Also, it is evident that the WORST ensemble levels out at a lower prediction accuracy than the BEST and SELECTED ensembles as the number of training examples grows and the rate of increase has stabilized.

Experiment B shows *no significant difference in the prediction accuracy between BEST and SELECTED*. Thus, our results do *not* confirm that prediction accuracy is improved by applying ensemble selections with a diverse set of individual classifiers. One possible explanation for this result is that the variation among the individual classifiers in the BEST ensemble already is enough to obtain a high prediction accuracy. There is clear evidence that the WORST ensemble performs worse than BEST and SELECTED. As a consequence, *simply using SG does not guarantee good results – the ensemble selection plays an important role*.

6.2 Experiment C: Learning Curves

In Experiment C, we study how consistent the learning curves for SG are across different industrial projects. Figure 8 depicts the learning curves for the five systems. As presented in Section 5.5, the BEST and SELECTED ensembles yield similar prediction accuracy, i.e., the learning curves in Figure 8 (a) and (c) are hard to distinguish by the naked eye. Also, while there are performance differences across the systems, the learning curves for all five systems follow the same general pattern: the learning curves appear to follow a roughly logarithmic form proportional to the size of the training set, but with different minimum and maximum values.

An observation of practical value is that the learning curves tend to flatten out within the range of 1,600 to 3,200 training examples for all five systems. We refer to this breakpoint as where the graph has the maximum curvature, i.e., the point on the graph where the tangent curve is the most sensitive to moving the point to nearby points. For our study, it is sufficient to simply determine the breakpoint by looking at Figure 8, comparable to applying the "elbow method" to find a suitable number of clusters in unsupervised learning [447]. Our results suggest that at least 2,000 training examples should be used when training a classifier for bug assignment.

We answer RQ3 as follows: *the learning curves for the five systems have different minimum and maximum values, but display similar shape and all flatten out at roughly 2,000 training examples. There is a clear difference between projects.*

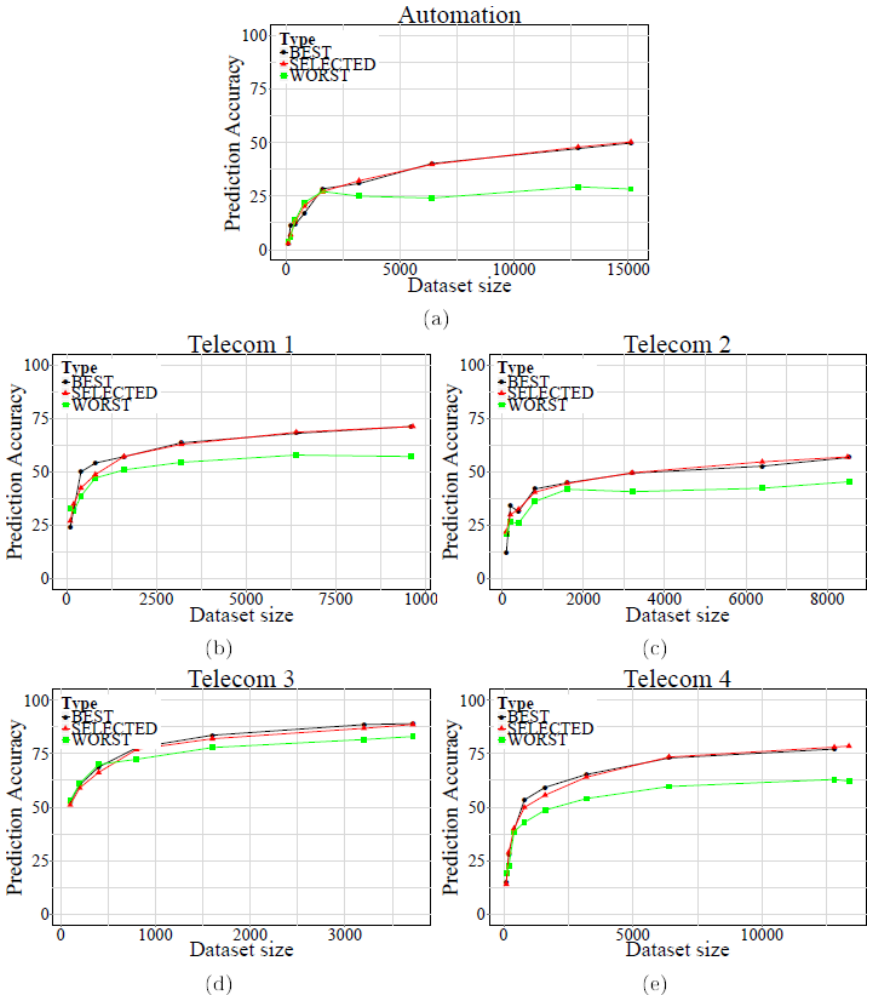


Figure 7: Comparison of BEST (black, circle), SELECTED (red, triangle) and WORST (green, square) classifier ensemble.

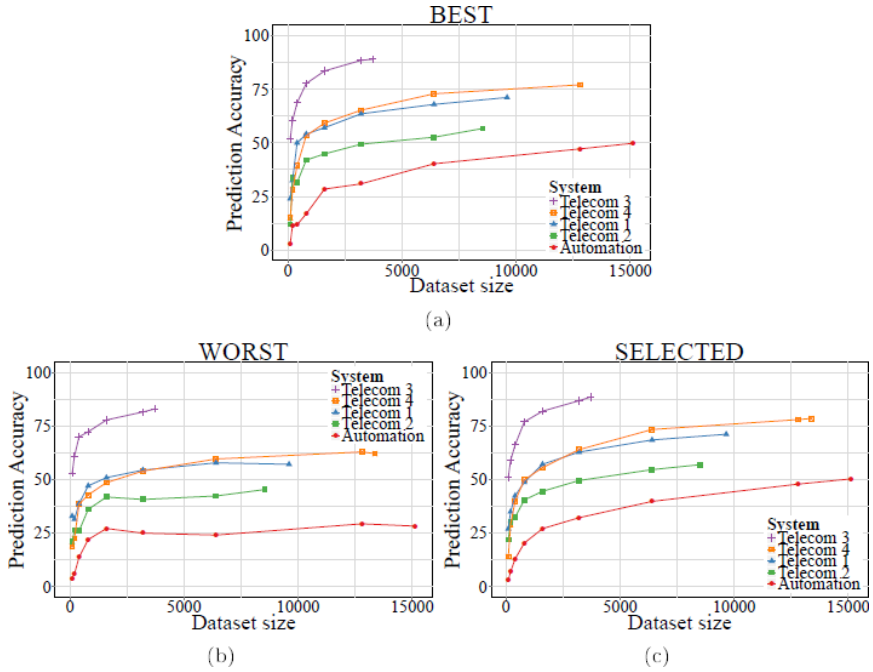


Figure 8: Prediction accuracy for the different systems using the BEST (a) WORST (b) and SELECTED (c) individual classifiers under Stacking

6.3 Experiment D: Sliding Time Window

Experiment D targets how the time locality of the training data affects the prediction accuracy (RQ4). Better understanding of this aspect helps deciding the required frequency of retraining the classification model. Figure 9 show the prediction accuracy of using SG with the BEST ensemble, following the experimental design described in Section 5.5. The X axes denote the difference in time, measured in days, between the start of the training set and the start of the test set. The figures also depict an exponential best fit.

For all datasets, the prediction accuracy decreases as older training sets are used. The effect is statistically significant for all datasets at a 5% level. We observe the highest effects on Telecom 1 and Telecom 4, where the prediction accuracy is halved after roughly 500 days. For Telecom 1 the prediction accuracy is 50% using the most recent training data, and it drops to about 25% when the training data is 500 days old. The results for Telecom 4 are analogous, with the precision accuracy dropping from about 40% to 15% in 500 days.

For three of the datasets the decrease in prediction accuracy is less clear. For Automation, the prediction accuracy decreases from about 14% to 7% in 1,000 days, and Telecom 3, the smallest dataset, from 55% to 30% in 1,000 days. For Telecom 2 the decrease in prediction accuracy is even smaller, and thus unlikely to be of practical significance when deploying a solution in industry.

A partial answer to RQ4 is: *more recent training data yields higher prediction accuracy when using SG for bug assignment*.

6.4 Experiment E: Cumulative Time Window

Experiment E addresses the same RQ as Experiment D, namely how the time locality of the training data affects the prediction accuracy (RQ4). However, instead of evaluating the effect using a fixed size sliding window of training examples, we use a cumulatively increasing training set. As such, Experiment E also evaluates how many training examples SG requires to perform accurate classifications. Experiment E shows the prediction accuracy that SG would have achieved at different points in time if deployed in the five projects under study.

Figure 10 plot the results from the cumulated time locality evaluation using SG with BEST ensembles. The curve represents the prediction accuracy (as fitted by a local regression spline) with the standard error for the mean of the prediction accuracy in the shaded region. The maximum prediction accuracy (as fitted by the regression spline) is indicated with a vertical line. The vertical line represents the cumulated ideal number of training points for the respective datasets. Adding more bug reports further back in time worsens the prediction accuracy. The number of samples (1589) and the prediction accuracy (16.41 %) for the maximum prediction accuracy is indicated with a text label (x = 1589 Y = 16.41 for the Automation system). The number of evaluations run with the calculated training set and test

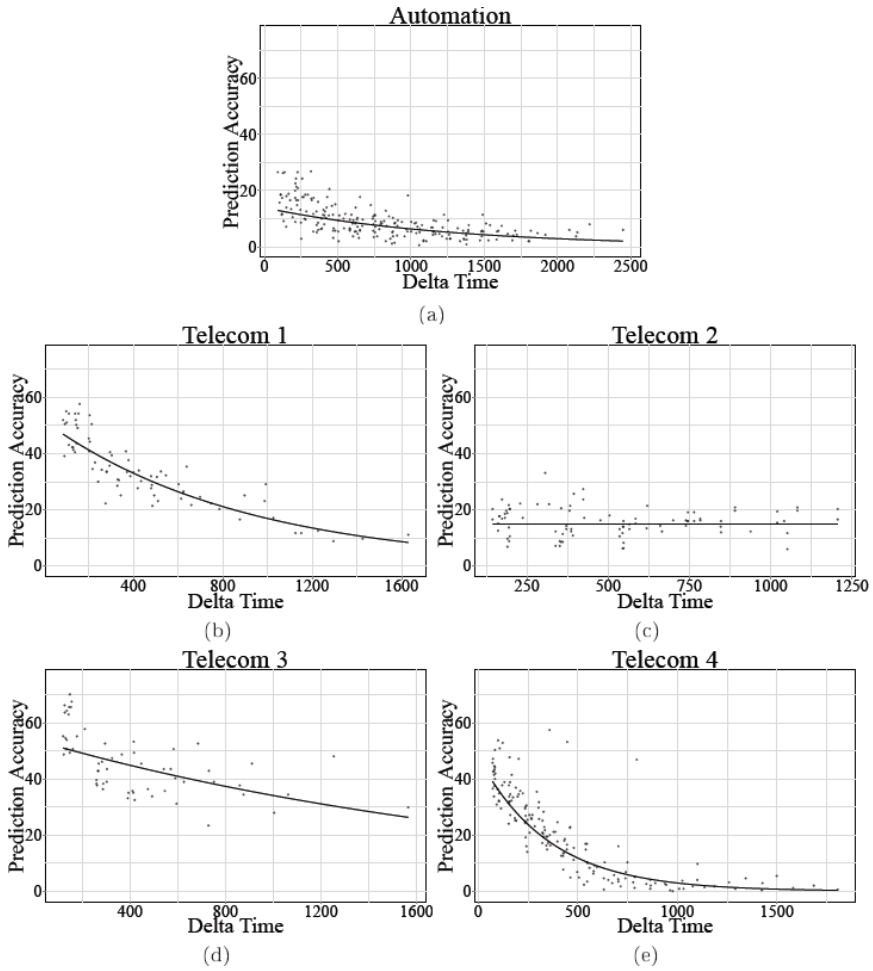


Figure 9: Prediction accuracy for the datasets Automation (a) and Telecom 1-4 (b-e) using the BEST ensemble when the time locality of the training set is varied. Delta time is the difference in time, measured in days, between the start of the training set and the start of the test set. For Automation and Telecom 1,2, and 4 the training sets contain 1,400 examples, and the test set 350 examples. For Telecom 3, the training set contains 619 examples and the test set 154 examples.

set sizes in each run is indicated in the upper right corner of the figure with the text "Total no. Evals".

For all datasets in Figures 10, except Telecom 3, the prediction accuracy increases when more training data is cumulatively added until a point where they reach a "hump" where the prediction accuracy reaches a maximum. This is followed by declining prediction accuracy as more (older) training data is cumulatively added. For Automation, Telecom 1, and Telecom 4, we achieve the maximum prediction accuracy when using about 1,600 training examples. For Telecom 2 the maximum appears already at 1,332 training examples. For Telecom 3 on the other hand, the curve is monotonically increasing, i.e., the prediction accuracy is consistently increasing as we add more training data. This is likely a special case for this dataset where we have not yet reached the point in the project where old data starts to introduce noise rather than helping the prediction.

Also related to RQ4 is our observation: *there is a balancing effect between adding more training examples and using older training examples. As a consequence, prediction accuracy does not necessary improve when training sets gets larger.*

7 Threats to Validity

We designed our experiment to minimize the threats to validity, but still a number of decisions that might influence our results had to be made. We discuss the main validity threats to our study with respect to construct validity, internal validity, external validity, and conclusion validity [477].

7.1 Construct Validity

Construct validity involves whether our experiment measures the construct we study. Our aim is to measure how well automated bug assignment performs. We measure this using *prediction accuracy*, the basic measure for performance evaluations of classification experiments. As an alternative measure of the classification performance, we could have complemented the results with average F-measures to also illustrate type I and type II errors. However, to keep the presentation and interpretation of results simple, we decided to consistently restrict our results to contain only prediction accuracy.

The Weka framework does not allow evaluations with classes encountered in the test set that do not exist in the training set. For Experiments A-C (all based on cross-validation) Weka automatically *harmonizes* the two sets by ensuring the availability of all classes in both sets. However, for the time-sorted evaluations performed in Experiment D and E, we do not use the cross-validation infrastructure provided by Weka. Instead, we have to perform the harmonization manually. To simplify the experimental design and be conservative in our performance estimates in Experiment D and E, we always consider all teams present in the full

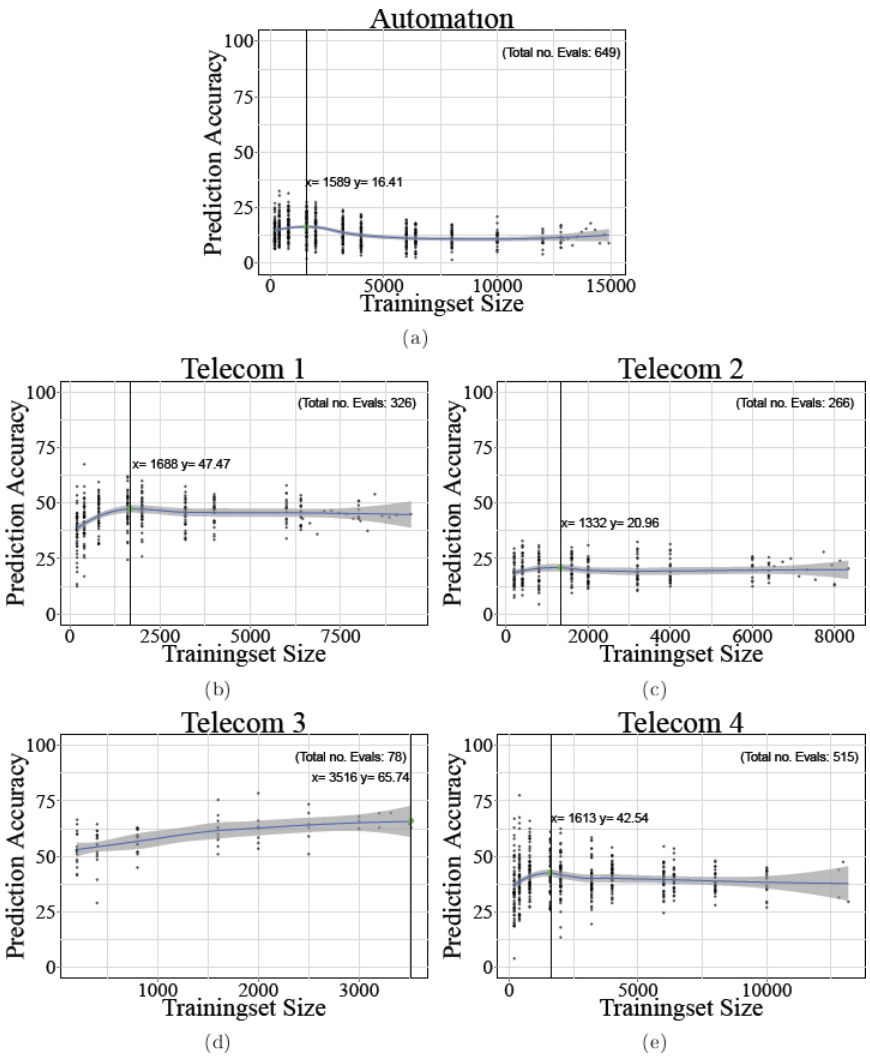


Figure 10: Prediction accuracy using cumulatively (further back in time) larger training sets. The curve represents the prediction accuracy (fitted by a local regression spline) with the standard error for the mean in the shaded region. The maximum prediction accuracy (as fitted by the regression spline) is indicated with a vertical line. The number of samples (1589) and the accuracy (16.41 %) for the maximum prediction accuracy is indicated with a text label (x = 1589 Y = 16.41 for the Automation system). The number of evaluations done is indicated in the upper right corner of the figure (Total no. Evals).

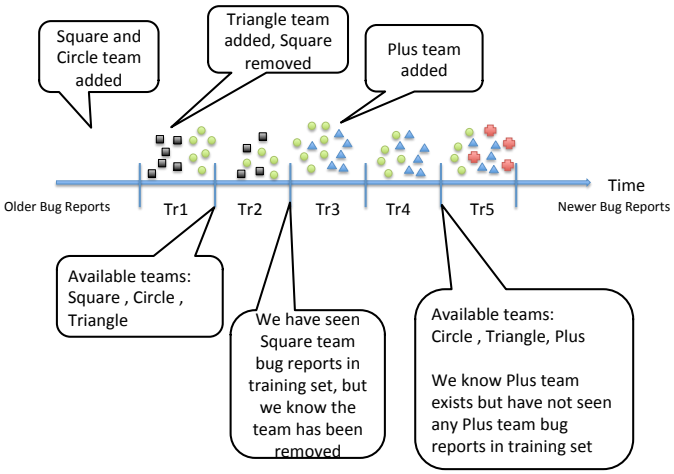


Figure 11: Team dynamics and BTS structure changes will require dynamic re-training of the prediction system. Teams are constantly added and removed during development so a prediction system must be adapted to keep these aspects in mind. These are all aspects external to pure ML techniques, but important for industry deployment.

dataset as possible classification results, i.e., regardless of whether the teams are present in the training and test sets of a specific run. This is a conservative design choice since it causes many more possible alternative classes available for classification, making it a harder problem. In practice, the team structure in an organization is dynamic. In the projects under study, teams are added, renamed, merged, and removed over the years as illustrated in Figure 11. While the current team structure would be known at any given time in a real project, we do not use any such information in our experimental runs. Thus, there is a potential that the prediction accuracy of a deployed tool using SG could be higher.

In the projects we study, the teams are not entirely disjunct. Individual developers might be members of several teams, teams might be sub-teams of other teams, and certain teams are created for specific tasks during limited periods of time. Thus, as some teams overlap, more than one team assignment could be correct for a bug report. Furthermore, the correctness of a team assignment is not a binary decision in real life. Teams might be hierarchically structured and there are dependencies between teams. An incorrect bug assignment might thus

be anything from totally wrong (e.g., assigning a bug report describing embedded memory management to a GUI team) to just as good as the team assignment stored in the BTS. Again, our evaluation is conservative as we consider everything not assigned to the same team as in the BTS as incorrect.

7.2 Internal Validity

Internal validity concern inferences regarding casual relationships. We aim to understand how SG performs compared to individual classifiers, and how its prediction accuracy is affected by different training configurations. Our experimental design addresses threats to internal validity by controlling the independent variables in turn. Still, there are a number of possibly confounding factors.

We conduct the same preprocessing for all classification runs. It is possible that some of the classifiers studied perform better for our specific choice of preprocessing actions than others. On the other hand, we conduct nothing but standard preprocessing (i.e., lower casing and standard stop word removal), likely to be conducted in most settings.

We use default configurations of all individual classifiers studied. While most classifiers are highly configurable, we do not perform any tuning. Instead, we consistently use the default configurations provided by Weka. The default configurations for some classifiers might be favorable for team assignment and others might underperform. Furthermore, we evaluated only one single level-1 classifier in SG, also using the default configuration. However, Wolpert argues that a simple level-1 classifier should be sufficient [478].

7.3 External Validity

External validity reflect the generalizability of our results. We study five large datasets containing thousands of bug reports from proprietary development projects. All datasets originate from development of different systems, including middleware, client-server solutions, a compiler, and communication protocols. However, while the datasets all are large, they originate from only two different companies. Furthermore, while the two companies work in different domains, i.e., automation and telecommunication, both are mainly concerned with development of embedded systems. To generalize to other domains such as application development, replications using other datasets are required.

The fraction of bug reports originating from customers is relatively low in all five datasets under study. In development contexts where end users submit more of the bug reports, different natural language descriptions and information content might be used. This might have an impact on the performance of SG for team assignment. However, as opposed to most previous work, we focus on proprietary development projects using closed BTSs.

We filtered the five datasets to contain only bug reports actually describing defects in the production software. It is possible that a BTS is used for other types of issues, as is the case in Company Automation, e.g., document changes, change requests, and changes to internal test code. We have not studied how well SG generalizes for these more generic types of issues. On the other hand, we assume that the most challenging team assignment involves defects in the production software where the location in source code and the related documentation are unknown.

7.4 Conclusion Validity

Conclusion validity is the degree to which conclusions we reach about relationships in our data are reasonable. For Experiment A, B, and C we use 10-fold cross validation as conventional in machine learning evaluations. However, as argued by Rao *et al.* [388], evaluations should also be performed using a *sequestered* test set. We accomplish this by performing Experiment D and E on separate training and test sets. Moreover, we evaluate the performance in several runs as described in Section 5.5.

The results from 10-fold cross-validation (using stratified sampling) and the evaluations conducted using defect reports submitted by submission date are different. Cross validation yields higher prediction accuracy, in line with warnings from previous research [280, 388]. To confirm the better results when using cross validation, we validated the results using RapidMiner [222] for the two datasets Automation and Telecom 4. We trained a Naïve Bayes classifier for Automation and an SVM classifier for Telecom 4 and observed similar differences between evaluations using 10-fold cross validation and a sorted dataset.

8 Discussion

This section contains a discussion of the results from our experiments in the context of our overall goal: to support bug assignment in large proprietary development projects using state-of-the-art ML. Section 8.1 synthesizes the results related to our RQs and discusses the outcome in relation to previous work. Finally, Section 8.2 reports important experiences from running our experiments and advice on industrial adaptation.

8.1 Stacked Generalization in the Light of Previous Work

We conduct a large evaluation of using SG for bug assignment, extending our previous work [255]. Our results show that *SG* (i.e., combining several classifiers in an ensemble learner) *can yield a higher prediction accuracy than using individual general purpose classifiers* (RQ1, ExpA). The results are in line with findings in general ML research [287]. However, we show that simply relying on SG is not

enough to ensure good results; some *care must be taken when doing the ensemble selection* (RQ2, ExpB). On the other hand, our results confirm the thesis by Wolpert, i.e., that SG should on average perform better than the individual classifiers included in the ensemble [478].

We present the first study on bug assignment containing 10,000s of bug reports collected from different proprietary development projects. Previous work has instead focused on bug reports from OSS development projects, e.g., Eclipse and Firefox, as presented in Section 3. A fundamental difference is that while bug assignment in OSS projects typically deal with individual developers, we instead assign bug reports to development teams. As this results in a more coarse assignment granularity, i.e., our output is a set of developers, one could argue that we target a less challenging problem.

We achieve prediction accuracy between 50% and 85% for our five systems using cross-validations, and between 15% and 65% for time-sorted evaluations. Thus, our work on bug team assignment does not display higher prediction accuracy than previous work on automated bug assignment to individuals, but is similar to what has been summarized in Table 2. Consequently, we show that *automated proprietary bug assignment, on a team level, can correctly classify the same fraction of bug reports as what has been reported for bug assignment to individual developers in OSS projects*. Bug assignment to teams does not appear to be easier than individual assignment, at least not when considering only the top candidate team presented by the ML system.

Cross-validation consistently yielded higher prediction accuracy than conducting more realistic evaluations on bug reports sorted by the submission date. The dangers of cross-validation have been highlighted in ML evaluation before [388], and it is a good practice to complement cross-validation with a sequestered test set. Our experiments show that evaluations on bug assignment can not rely on cross-validation alone. Several factors can cause the lower prediction accuracy for the time sorted evaluations. First, cross-validation assumes that the bug reports are independent with no distributional differences between the training and test sets [26]. Bug reports have a natural temporal ordering, and our results suggest that the dependence among individual bug reports can not be neglected. Second, we used stratified sampling in the cross-validation, but not in the time sorted evaluations. Stratification means that the team distributions in the training sets and test sets are the same, which could improve the results in cross-validation. Third, as we perform manual harmonization of the classes in the time sorted evaluation (see Section 7), all teams are always possible classifications. In cross-validation, Weka performs the harmonization just for the teams involved in the specific run, resulting in fewer available teams and possibly a higher prediction accuracy.

8.2 Lessons Learned and Industrial Adoption

Two findings from our study will have practical impact on the deployment of our approach in industrial practice. First, we studied how large the training set needs to be for SG to reach its potential. The learning curves from 10-fold cross-validation show that larger training set are consistently better, but the improvement rate decreases after about 2,000 training examples. The point with the maximum curvature, similar to an elbow point [447] as used in cluster analysis, appears in the same region for all five systems. As a result, we suggest, as a rule of thumb, that *at least 2,000 training examples should be used when using SG for automated bug assignment* (RQ3, ExpC).

The second important finding of practical significance relates to how often an ML system for bug assignment needs to be retrained. For all but one dataset, our results show a clear decay of prediction accuracy as we use older training data. For two datasets the decay appears exponential, and for two datasets the decay is linear. Our conclusion is that the time locality of the training data is important to get a high prediction accuracy, i.e., *SG for bug assignment is likely to achieve a higher prediction accuracy if trained on recent bug reports* (RQ4, ExpD). Bhattacharya *et al.* [55] recently made the same observation for automated bug assignment using large datasets from the development of Eclipse and Mozilla projects.

Finding the right time to retrain SG appears to be a challenge, as we want to find the best balance between using many training examples and restricting the training set to consist of recent data. In Experiment E, our last experiment, we try several different cumulatively increasing training sets at multiple points in time. This experimental setup mimics realistic use of SG for bug assignment, trained on different amounts of previous bug reports. We show that for four of our datasets, *the positive effect of using larger training sets is nullified by the negative effect of adding old training examples*. Only for one dataset it appears meaningful to keep as much old data as possible.

When deployed in industrial practice, we recommend that *the prediction accuracy of automated bug assignment should be continuously monitored to identify when it starts to deteriorate*. For four of our datasets, cumulatively increasing the amount of training data is beneficial at first (see Fig. 10), but then SG reaches a maximum prediction accuracy. For all but one dataset, the prediction accuracy starts to decay even before reaching the 2,000 training examples recommended based on the results from Experiment C. Furthermore, we stress that attention should be paid to alterations of the prediction accuracy when significant changes to either the development process or the actual software product are made. Changes to the team structure and the BTS clearly indicate that SG should be retrained, but also process changes, new tools in the organization, and changes to the product can have an impact on the attributes used for automated bug assignment. In practice, the monitoring of the prediction accuracy could be accomplished by measuring the amount of bug tossing taking place after the automated bug assignment has taken

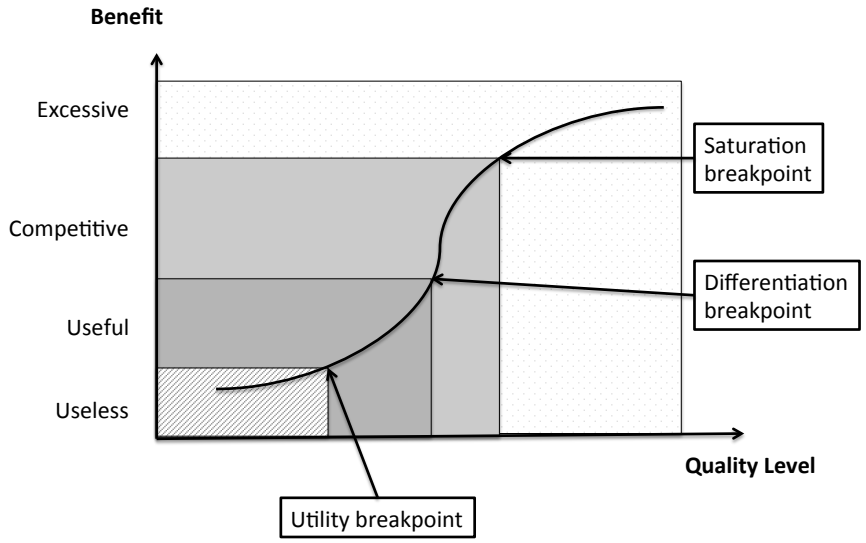


Figure 12: Perceived benefit vs. prediction accuracy. The figure shows two breakpoints and the current prediction accuracy of human analysts. Adapted from Regnell *et al.* [390].

place.

While we can measure the prediction accuracy of SG for bug assignment, it is not clear what this means for practical purposes. How accurate do the classifications have to be before developers start recognizing its value? Regnell *et al.* [390] describe quality, in our case the prediction accuracy of automated bug assignment, as continuous and non-linear. Figure 12 shows what this means for bug assignment. *The perceived usefulness of SG is on a sliding scale with specific breakpoints*. The utility breakpoint represents when developers start considering automated bug assignment useful, any prediction accuracy below this level is useless. The saturation breakpoint indicates where increased prediction accuracy has no practical significance to developers. Figure 12 also displays the prediction accuracy of human analysts between the breakpoints. We argue that automated bug assignment does not have to reach the human accuracy to be perceived useful, as the automated process is much faster than the manual process. Our early evaluations indicate that the prediction accuracy of SG in Company Telecom is in line with the manual process [255]. Even though this study also used 10-fold cross-validation, which we have seen can give overly optimistic estimates in this context; we believe that our prototype has passed the utility breakpoint before we have started any context specific tuning of SG.

When we implement automated bug assignment in an industrial tool, we plan

to present a handful of candidate teams to the user for each bug report under investigation. While we could automatically assign the bug to the first candidate, our first step is to provide decision support to the CCB. Considering the automation levels defined by Parasuraman *et al.* [367], this reflects an increase in automation from level 0 to level 3: "narrowing the selection down to a few". By presenting a limited selection of teams, possibly together with a measure of the confidence level from SG, an experienced developer can quickly choose the best target for the bug assignment. Note that the experimental design we used in the evaluations in this study are stricter as we only considered one single team assignment per bug report. Another recommendation is to plan from the start to run the new ML-based system in parallel with the old way of working, to evaluate if the prediction accuracy is good enough for a complete roll over to a process supported by ML.

Furthermore, we must develop the tool to present the candidate teams to the user in a suitable manner. Murphy-Hill and Murphy presents several factors that affects how users perceive recommendations provided by software engineering tools [346]. The two most important factors for our tool are transparency and the related aspect assessability. A developer must be able to see why our tool suggests assigning a bug report to a specific team, i.e., *the rationale leading to the SG classification must be transparent*. Also, our tool should *support developers in assessing the correctness of a suggested assignment*. We aim to achieve this by enabling interaction with the output, e.g., browsing previous bug assignments, opening detailed bug information, and comparing bug reports.

9 Conclusions and Future Work

We conduct the first evaluation of automated bug assignment using large amounts of bug reports, collected from proprietary software development projects. Using an ensemble learner, Stacked Generalization (SG), we train an ML system on historical bug reports from five different projects in two different, large companies. We show that *SG consistently outperforms individual classifiers with regard to prediction accuracy* even though the improvements are sometimes marginal (RQ1). Moreover, our results suggest that it is *worthwhile to strive for a diverse set of individual classifiers in the ensemble* (RQ2), consistent with recommendations in the general ML research field. Our results show that SG, with feasible ensemble selection, can reach prediction accuracies of 50% to 90% for the different systems, in line with the prediction accuracy of the current manual process. We also briefly study the relative value of textual vs. non-textual features, and conclude that the most promising results are obtained when combining both in SG. In future work we plan to improve the textual features with more advanced text modeling techniques such as Topic Modeling (LDA).

We study the SG learning curves for the five systems (RQ3), using 10-fold cross-validation. The learning curves for all five datasets studied display similar

behaviour, thus we present an empirically based rule-of-thumb: *when training SG for automated bug assignment, aim for at least 2,000 bug reports in the training set*. Using time-sorted bug reports in the training and test sets we show that *the prediction accuracy decays as older training data is used* (RQ4). Consequently, we show that *the benefit of adding more bug reports in the training set is nullified by the disadvantage of training the system on less recent data*. Our conclusion is that *any ML system used for automated bug assignment should be continuously monitored to detect decreases in prediction accuracy*.

Our results confirm previous claims that *relying only on K-fold cross-validation is not enough to evaluate automated bug assignment*. We achieve higher prediction accuracy when performing 10-fold cross-validation with stratification than when analyzing bug reports sorted by the submission date. The differences we observe are likely to be of practical significance, thus it is important to report evaluations also using sorted data, i.e., mimicking a realistic inflow of bug reports. Several authors have proposed modifications to cross-validation to allow evaluations on dependent data, e.g., *h*-block cross-validation [90]. Future work could try this for bug assignment evaluation, which means reducing the training set by removing *h* observations preceding and following the observations in the test set.

When deploying automated bug assignment in industry, we plan to present more than one candidate development team to the user of the ML system. By presenting a ranked list of teams, along with rationales of our suggestions, an experienced member of the CCB should be able to use the tool as decision support to select the most appropriate team assignment. Our current evaluation does not take this into account, as we only measure the correctness of the very first candidate. Future work could extend this evaluation by evaluating lists of candidates, opening up for measures from the information retrieval field, e.g., mean average precision and normalized discounted cumulative gain. Finally, to properly evaluate how ML can support bug assignment in industry, the research community needs to conduct industrial case studies in organizations using the approach. In particular, it is not clear how high the prediction accuracy needs to be before organizations perceive the system to be "good enough".

Future work could be directed toward improving our approach to automated bug assignment. A number of studies in the past show that tools specialized for bug assignment in a particular project can outperform general purpose classifiers [444, 482, 484]. It would be possible for us to explore if any enhancements proposed in previous work could improve the accuracy of SG, e.g., topic models, social network analysis, or mining the commit history of source code repositories. Also, we could further investigate if any particular characteristics of team assignment in proprietary projects could be used to improve automated bug assignment, i.e., characteristics that do not apply to OSS projects.

Another direction for future enhancements of our approach could explore how to adapt bug assignment based on the developers' current work load in the organization. The current solution simply aims to assign a bug report to development

teams that worked on similar bug reports in the past. Another option would to optimize the resolution times of bug reports by assigning bugs to the team most likely to close them fast. For many bug reports, more than one team is able to resolve the issue involved, especially in organizations with a dedicated strategy for shared code ownership. Future work could explore the feature engineering required for SG to cover this aspect. Yet another possible path for future work, made possible by the large amount of industrial data we have collected, would be to conduct comparative studies of bug reports from OSS and proprietary projects, similar to what Robinson and Francis reported for source code [404].

Acknowledgements

This work was supported in part by the Industrial Excellence Center EASE – Embedded Applications Software Engineering[8].

[8] http://ease.cs.lth.se

CHAPTER IV

SUPPORTING CHANGE IMPACT ANALYSIS USING A RECOMMENDATION SYSTEM: AN INDUSTRIAL CASE STUDY IN A SAFETY-CRITICAL CONTEXT

Abstract

Change Impact Analysis (CIA) during software evolution of safety-critical systems is a fundamental but labor-intensive task. Several authors have proposed tool support for CIA, but very few have been evaluated in industry. We present ImpRec, a Recommendation System for Software Engineering (RSSE), tailored for CIA at an automation company. Building on research from assisted tracing using information retrieval solutions, and mining software repositories, ImpRec recommends development artifacts potentially impacted when resolving incoming issue reports. In contrast to previous work on automated CIA, our approach explicitly targets development artifacts that are not source code. We evaluate ImpRec in a two-phase industrial case study. First, we measure the correctness of ImpRec's recommendations by simulating the historical inflow of 12 years worth of issue reports in the company. Second, we assess the utility of working with ImpRec by deploying the RSSE in two development teams. Our results suggest that ImpRec presents about 40% of the true impact among the top-10 recommendations. Furthermore, user log analysis indicates that ImpRec can support CIA in industry, and developers acknowledge the value of ImpRec in interviews. In conclusion, our findings show the potential of reusing traceability associated with developers'

past activities in an RSSE. However, more research is needed on how to retrain the tool once deployed, and how to adapt processes when new tools are introduced in safety-critical contexts.

Markus Borg, Krzysztof Wnuk, Björn Regnell, and Per Runeson *To be submitted*

1 Introduction

Large-scale software-intensive systems evolve for years. Several studies report that software evolution might last for over a decade, e.g., at Siemens [462], Ericsson [170], and ABB [5]. Long-term evolution results in complex legacy systems in which changes are known to be labor-intensive [68] and error-prone [103]. However, understanding how changes propagate in large systems is fundamental in software evolution. In an analysis of 14 recent catastrophic accidents caused by software failures, Wong *et al.* report inadequate change management as one of the main factors [480].

In safety-critical software engineering, formal Change Impact Analysis (CIA) is mandated in process standards such as the general standard IEC 61508 [239], IEC 61511 in the process industry sector [238], ISO 26262 in the automotive domain [242], DO-178C in avionics [382], and EN 50128 for the railway sector [174]. The standards state that a CIA must be conducted prior to any software change, but do not contain details on *how* the activity shall be practically conducted. Thus, the implementation of the CIA activity is specific to the development organization, and an important component of the safety case [268], i.e., the structured argument evaluated by external assessors to justify safety certification [350].

Several studies report that CIA is a tedious part of software maintenance and evolution, e.g., in process automation [71], in nuclear power [109], in the automotive industry [208], and in the aerospace domain [271]. Still, the level of CIA tool support in industry is low, as we show in a recent cross-domain survey [136]. The low level of CIA automation has also been reported in previous work by Lettner *et al.* [296]. Moreover, Li *et al.* report that most proposed CIA tools are restricted to source code impact, but also other artifacts are important, e.g., UML models [87], quality requirements [117], and regression test plans [363]. This paper focuses on filling this gap by explicitly focusing on *CIA of software artifacts that are not source code*, i.e., non-code artifacts.

Several researchers claim that semi-automated tracing applying Information Retrieval (IR) techniques can support CIA. A handful of controlled experiments with student subjects show favorable results [72, 128, 148]. Moreover, initial evidence from a small industrial case study in China suggests that IR-based tracing tools has the potential to support CIA [304]. However, still no tool implementing IR-based tracing has been evaluated in a large software development context [76].

Although scalability was identified as one of the eight "*grand challenges of traceability*" by CoEST[1], most previous evaluations on IR-based tracing scaled down the challenge to small datasets containing fewer than 500 artifacts [76]. Instead, scalability is often discussed as a threat to validity, and highlighted as important concerns for future work (see, e.g., De Lucia *et al.* [144] and Huffman Hayes *et al.* [232]). The work presented in this paper focuses on designing a scalable solution that matches industry needs. Our solution is based on analyzing large amounts of artifact relations, i.e., we leverage on the volume of information available, and evaluate the solution in a real world context.

We build on previous work on IR-based tracing tools, and enhance it with a semantic network mined from historical CIA data recorded in an issue tracker. Our combined approach is implemented in *ImpRec* [75], a Recommendation System for Software Engineering (RSSE) [402]. ImpRec is tailored for one of our industry partners active in process automation, although comprising general solution elements. The RSSE assists developers performing CIA as part of issue management by providing recommendations of potentially impacted artifacts. ImpRec identifies textually similar issue reports, and recommends candidate impact by analyzing a semantic network of development artifacts.

We evaluate ImpRec in a large-scale industrial case study using user-oriented evaluation. Robillard and Walker highlight the absence of user oriented evaluations in a recently published book on RSSEs. They claim that that many tools have been fully implemented, but "*much less energy has been devoted to research on the human aspects*" [402]. Our study responds to their call for in-depth evaluations, by reporting from a longitudinal *in situ* study of ImpRec deployed in two proprietary development teams. Also, we perform a static validation [201] of ImpRec, resulting in quantitative results reflecting common evaluation practice in both RSSE and traceability research.

The main parts of the paper are:

- A comprehensive background section on CIA, IR in software engineering, and RSSEs for change management (Section 2).
- A detailed context description that supports understanding and analytical generalization to other cases (Section 3). We also introduce several challenges experienced in state-of-practice CIA, establishing relevant directions for future inquiries.
- A presentation of ImpRec, a prototype RSSE for CIA in a safety-critical context (Section 4.4). ImpRec is unique in specifically targeting CIA for non-code artifacts.
- An in-depth case study, comprising both a quantitative static validation and a qualitative dynamic validation (Section 5). Thus, our study is designed to

[1] The Center of Excellence for Software Traceability, an organization of academics and practitioners with a mission to advance traceability research, education, and practice (www.coest.org).

assess both the *correctness* of the ImpRec output, and the *utility* of actually working with ImpRec.

- We show that ImpRec recommends about 40% of the true impact among the top-10 recommendations (Section 6.1). Also, developers report that ImpRec's current level of correctness can be helpful when conducting CIA (Section 6.2).
- A thorough discussion on threats to validity (Section 7), and implications for research and practice (Section 8).

2 Background and Related Work

This section provides a background on CIA in safety-critical contexts. Then we present related work on IR in software engineering, and the closest work on RSSEs for change management. We conclude the section by reporting our previous work on the topic.

2.1 Change Impact Analysis

A fundamental facilitator in the success of software is that software artifacts can be modified faster than most of its counterparts in other engineering disciplines. Unfortunately, understanding the impact of changes to complex software systems is tedious, typically avoided unless necessary [68], and instead mended by extensive regression testing [171]. In safety-critical software development however, to avoid unpredictable consequences, CIA is a mandated practice [174, 239, 242].

Bohner defines CIA as "*identifying the potential consequences of a change, or estimating what needs to be modified to accomplish a change*" [68]. Furthermore he describes CIA as a process that is iterative and discovery in nature, i.e., identifying change impact is a cognitive task that incrementally adds items to a candidate impact set. Furthermore, he discusses three main obstacles for CIA support in practice. First, *information size volume* - artifacts from many different sources must be related and analyzed. Second, *change semantics* - methods for describing change relationships are lacking. Third, *analysis methods* - methods for analyzing software dependencies and traceability have not been fully explored.

Manual work dominates CIA in industry. A recent survey with software engineers from different safety-critical fields confirmed that the level of automation in CIA is low and that CIA support is mainly available in the source code level [136]. Moreover, insufficient tool support was mentioned as the most important challenge in the state-of-practice. The survey also highlighted that practitioners typically work with only basic tool support, e.g., MS Word and MS Excel, and no dedicated CIA tools are widely used. The importance of research on improved CIA tools has also been highlighted by other researches, e.g., stated by Bohner: "*ever-increasing*

need to wade through volumes of software system information [...] *automated assistance is important to cut analysis time and improve the accuracy of software changes*" [68], and Lehnert: "*more attention should be paid on linking requirements, architectures, and code to enable comprehensive CIA*" [292].

Two recent literature reviews confirm that most research on CIA is limited to *impact on source code*. Lehnert created a taxonomy for research on CIA [293], partly based on Arnold and Bohner's early work on a framework for CIA comparison [27] and populated this taxonomy with 150 primary studies from a literature review [292]. The results show that only 13% of the studies address multiple artifact types, and 65% targets only source code impact. Li *et al.* also concluded, in a systematic literature review, that the source code focus dominates among CIA researchers [301]. Based on the 30 publications in the study, they identified that more research is needed to go beyond mere research prototype tools evaluated in lab environments, to evaluations of how deployed tools perform in the software maintenance context. Moreover, Li *et al.* explicitly mentioned studies on tool support that provides ranked lists of potentially impacted entities. Gethers *et al.* present another good overview of tool support for CIA on the source code level [192].

Some previous work on CIA has also targeted the process automation domain. Acharya and Robinson developed *Imp*, a tool that uses static program slicing to assist developers with accurate impact analyses [5]. Imp was evaluated on a large proprietary evolving software system, and this is the first reported large-scale evaluation of CIA. While the evaluation addresses a software system consisting of more than a million lines of code, thus matching the scale of the system we studied in this paper, the proposed solution is restricted to impact on source code.

Another study on CIA for software-intensive systems in the process automation domain is presented by Ghosh *et al.* [196]. The authors studied 33 historical software development projects to explore impact analysis triggered by requirement changes. Their focus is however not on what specific artifacts are impacted, but how much effort the changes require in total. Using a linear regression model, they predict the cost of changing requirements, and practitioners confirm the value of their approach.

Not all research effort on CIA is devoted to tool support. Stålhane *et al.* suggest supporting CIA by *process improvements*, and propose agile change impact analysis as part of SafeScrum [440]. Their work is based on the lack of practical CIA guidelines, and they present an approach to move from typical heavy upfront analysis to more incremental work, i.e., agile. The same authors also present a new structure of CIA reports tailored for software engineering in railway and the process industry [348]. Jonsson *et al.* also studied how to apply agile software development in the railway domain and the authors conclude that impact analysis can be problematic in highly iterative development contexts [253].

Some researchers studied organizational aspects of CIA. Kilpinen identified that experiential CIA is common in industry, i.e., scoping changes through inspec-

tion and engineering judgement [270]. She introduced "*the adapted rework cycle*" to improve the interplay between CIA and the design process, and demonstrates its feasibility using simulation in an aerospace company. In a case study at Ericsson in Sweden [409], Rovegård *et al.* mapped the CIA work conducted by 18 interviewees to decision making at three different organization levels: *strategic* (large scope, long term), *tactical* (mid-term, e.g., resource allocation) and *operative* (short term, technical details). Furthermore, they explore current challenges involved in state-of-practice CIA, and propose five explicit improvements to address them, including two that we employ in this paper: a) introducing a knowledge base of old CIA results, and b) introducing better tool support.

2.2 Information Retrieval in Software Engineering

Large software engineering projects constitute complex information landscapes of thousands of heterogenous artifacts. These artifacts constantly change, thus providing developers quick and concise access to information is pivotal. Various IR systems support developers, including both general purpose search tools and more specialized solutions. This section presents research on the two areas of IR in software engineering relevant to our work: *IR-based tracing* [76] and *duplicate detection of issue reports.* [410].

More than a hundred publications on using IR for traceability management have been published since year 2000. A recent book on traceability makes an attempt to unify the definitions [203]. According to this work, *traceability creation* is the activity of associating artifacts with trace links. *Trace capture* involves creation of trace links concurrently with the artifacts that they associate, and trace recovery implies establishing trace links after the artifacts have been created. Using IR in the traceability context means to link artifacts with highly similar textual content. We use the term *IR-based tracing tools* to refer to similarity-based tools for either trace recovery or trace capture.

Antoniol *et al.* were the first to express identification of trace links as an IR problem [18]. They used the Binary Independence Model (BIM) and the Vector Space Model (VSM) to generate candidate trace links between design and source code. Marcus and Maletic introduced Latent Semantic Indexing (LSI) to recover trace links between source code and natural language documentation [327]. Huffman Hayes *et al.* enhanced VSM-based trace recovery with relevance feedback. Their approach is clearly human-oriented and aims at at supporting V&V activities at NASA using their tool called *RETRO* [231]. De Lucia *et al.*'s research in this topic focuses on empirically evaluating LSI-based trace recovery in their document management system called *ADAMS* [144]. They advanced the front of empirical research on IR-based tracing by conducting a series of controlled experiments and case studies with student subjects. Cleland-Huang and colleagues have published several studies using probabilistic IR models for trace recovery, implemented in their tool *Poirot* [500]. Much of their work has focused on improving the accu-

racy of their tool by various enhancements, e.g., project glossaries and term-based methods.

We have recently published a systematic mapping study of IR-based trace recovery [76]. The majority of the proposed approaches reported in the identified 79 papers were evaluated on small sets of software artifacts, often originating from student projects [79]. Furthermore, we found that most evaluations of IR-based trace recovery do not involve humans, i.e., constitute *in silico* studies, and only one primary study was conducted as a case study in industry [304]. We have also reported that there appears to be little practical difference between different IR models [74], which corroborates previous observations by other researchers [72, 128, 360].

IR systems in software engineering were also applied for issue report duplicate detection. The results showed that, in large projects, the fraction of duplicates can be between 10-30% and might cause considerable extra effort during triage [22, 243, 410]. IR-based duplicate identification could substantially reduce this effort. We have previously summarized work on issue duplicate detection based on textual similarity, and concluded that duplicate detection is promising when the inflow of issue reports is large [75].

The RSSE we present in this paper builds on previous research on IR in software engineering. Similar to IR-based tracing tools, the goal of our tool is to identify trace links in databases containing a plethora of artifacts. Consequently, our work is also related to duplicate issue reports detection. We develop a tool intended to support developers with the inflow of issue reports. Most IR-based tracing tools on the other hand, address trace recovery rather than trace capture.

2.3 RSSEs for Change Management and CIA

RSSEs differ from other software engineering tools in three ways [402]. First, RSSEs primarily *provide information* (in contrast to tools such as static code analyzers and test automation frameworks). Second, RSSEs *estimate that a piece of information is valuable*, as opposed to "*fact generators*" (e.g., cross-reference tools and call browsers). Third, *the close relation to a specific task and context* distinguishes RSSEs from general search tools.

Several RSSEs support developers in navigating the complex information landscapes of evolving software systems. Čubranić *et al.* developed *Hipikat*, an RSSE to help newcomers resolve issue reports in large software projects [127]. Hipikat relies on Mining Software Repositories (MSR) [258] techniques to establish a project memory of previous experiences, including issue reports, commits, and emails. The project memory is stored as a semantic network in which relations are either explicit (e.g., email replies, commits implementing issue resolutions) or implicit (deduced based on textual similarities). Hipikat uses the semantic network to recommend potentially related information items to developers, e.g., source code, issue reports, and documentation.

Čubranić *et al.* implemented Hipikat in the context of the Eclipse OSS development, and present both an *in silico* evaluation (i.e., a computer experiment) as well as an *in vitro* user study (i.e., in a controlled lab environment). Both evaluations are mainly qualitative, and assess how helpful Hipikat is when resolving a new issue report. The IR evaluation reports an average precision and recall of 0.11 and 0.65, respectively, for a small set of 20 closed issue reports, comparing the source code files recommended by Hipikat and the actually modified source code files (the gold standard). The authors also qualitatively analyzed the Hipikat output. Čubranić *et al.* also involved four experienced Eclipse developers and eight experienced developers new to the Eclipse project to resolve (the same) two issue reports in a controlled environment. The results showed that Hipikat helps newcomers perform comparably to more knowledgeable developers, i.e., to implement high quality corrective fixes under time pressure.

Another RSSE is *eRose* (originally Rose) developed by Zimmermann *et al.* [497]. Using association rule mining in version control systems, eRose detects source code files that tend to change together, and delivers recommendations accordingly to support change management. Using eRose in a project brings two advantages: 1) improved navigation through the source code, and 2) prevention of omission errors at commit time. Zimmermann *et al.* used simulation [466] to evaluate three usage scenarios supported by eRose, including one scenario that resembles our ImpRec approach: recommending source code files that are likely to change given a source code file known to be modified. Using sequestered training and test sets from eight OSS projects, with the test sets in total comprising more than 8,000 commits, the top-10 recommendations from eRose contained 33% of the true changes. In 34% of the cases eRose does not provide any recommendations. On the other hand, when eRose indeed provides recommendations, a correct source code file is presented among the top-3 in 70% of the cases.

Ying *et al.* used association rule mining in version control systems to predict source code changes involved in software maintenance [486]. In line with the work by Zimmermann *et al.*, the authors evaluated their approach *in silico* using simulation on two large OSS projects, Eclipse and Mozilla, and they also investigated recommendations for additional source code impact when knowing for certain that one file was modified. The training and the test sets were sequestered in time and contained more than 500,000 revisions of more than 50,000 source code files. For Mozilla, precision is roughly 0.5 at recall 0.25, and for Eclipse, precision is 0.3 at recall 0.15. Ying *et al.* argues that while the precision and recall obtained were not high, the recommendations can still reveal valuable dependencies. They also introduced an "*interestingness value*" to assess how meaningful a recommendation is, but it is only applicable for source code.

Some researchers have explicitly mentioned CIA support as a promising use case for their RSSEs. Canfora and Cerulo developed *Jimpa* [92], an RSSE that uses MSR techniques to utilize past impact when conducting new CIAs. Using textual similarity calculations between issue reports and commit messages, Jimpa

recommends files that are likely to be impacted to resolve an issue. The authors evaluated Jimpa *in silico* using leave-one-out cross-validation on three medium-sized OSS projects containing in total 1,377 closed issue reports. Top-10 recommendations for the three OSS projects result in the following P-R pairs: 0.20-0.40, 0.05-0.20, and 0.15-0.90, and the authors consider the outcome promising.

Gethers *et al.* developed *ImpactMiner*, a tool that combines textual similarity analysis, dynamic execution tracing, and MSR techniques [192]. ImpactMiner mines evolutionary co-changes from the version control system, and uses the search engine library *Apache Lucene* to index the source code and to enable feature location. Furthermore, ImpactMiner can compare stack traces and present recommendations based on the co-changes. Gethers *et al.* present an *in silico* evaluation on four medium-sized OSS projects using a selected set of in total 181 closed issue reports. For the four OSS projects, the top-10 recommendations correspond roughly to precision 0.10-0.15 and recall 0.20-0.35.

2.4 Contribution in Relation to Previous Work

This section presents how we combine our previous research results and positions our contribution to related work.

Evolution of ImpRec Based on Previous Work

ImpRec is the result of an extensive research effort over three years that incorporate relevant previous results. In previous work, we have developed *ReqSimile*, a tool for consolidation of natural language requirements from multiple sources [353]. Positive results in identifying similar requirements with the help of ReqSimile laid the foundation for the identification of related issue reports in ImpRec. Moverover, previous positive results in using IR techniques at Sony Mobile Communications [410] also encouraged us for further research efforts. Next, we evaluated using the IR library Apache Lucene for duplicate detection in the Android issue tracker [78], obtaining promising results. Finally, we have conducted a systematic mapping of IR-based trace recovery [76], thoroughly investigating IR techniques applied and the quality of the empirical evaluations. We also conducted an initial *in vitro* experiment on IR-based trace recovery for CIA [72], whose results encouraged us to pursue further research. Finally, the use of networked structures was piloted in our previous work on the Android issue tracker and in a proprietary system [73], and we have proposed to support CIA by reusing traceability captured in an issue tracker [71].

ImpRec packages our previous research efforts in IR as an RSSE for CIA. While the technical implementation details were reported in a recently published book chapter [75], this paper brings significant empirical evidence from a large-scale industrial case study. Section 4 presents an overview of the approach implemented in ImpRec.

ImpRec in Relation to Previous Research

The development of ImpRec was influenced and inspired by several other studies. Among them, Čubranić *et al.*'s studies on Hipikat [126, 127] were central in making key decision about the ImpRec development and architecture. However, while Hipikat is intended to support project newcomers navigate OSS projects, ImpRec is instead tailored to support CIA in a specific industrial context. ImpRec implements several approaches presented in Hipikat, including: 1) a semantic network of artifacts and relations, 2) mining software repositories as the method to capture important relations from the project history, and 3) creation of links between issue reports based on textual similarity. For a thorough comparison of the internals of Hipikat and ImpRec, we refer to our previous book chapter [75].

Several studies proposed history mining to identify artifacts that tend to change together [92, 192, 486, 497]. Especially the studies by Canfora and Cerulo [92] and Gether *et al.* [192] use techniques similar to ours, as they combine mining with IR techniques. However, while these four studies address CIA, they solely focus on source code [292]. Our work aims to break new grounds by targeting CIA of non-code artifacts.

While ImpRec combines existing techniques in novel ways, the main contribution of this study comes from the empirical evaluation. Several RSSEs have been fully implemented, but very few have been evaluated *in situ*, i.e., with real developers using the RSSE as part of their normal work in a project [402]. While simulating the operation of an RSSE can provide the correctness of the recommendations, it does not reveal anything about the users' reactions. Two previous RSSE evaluations provide strength of evidence [166], in terms of level of realism, matching our study: Čubranić *et al.*'s study on Hipikat [127] and Kersten and Murphy's evaluation of MyLyn [269]. However, our study is unique in its combination of a thorough *in silico* evaluation and an *in situ* study in industry.

Compared to the Hipikat evaluation [127], our study on ImpRec is more thorough in three ways. First, the *in silico* evaluation of Hipikat used only 20 issue reports, randomly selected from a set of 215 feasible issue reports. We instead use 596 issue reports in our test set, and assure that they they are sequestered from the test set as recommended by Rao *et al.* [388]. Second, our test set contains all issue reports with CIA reports in our dataset, instead of a sample (as was also the case in the evaluations by Zimmermann *et al.* [497] and Ying *et al.* [486]). Third, while the number of participants in our user study is in line with the Hipikat study, we designed a longitudinal study *in situ* instead of a one-shot *in vitro* task in the lab.

The ImpRec evaluation presented in this paper shares several aspects of the MyLyn evaluation by Kersten and Murphy [269]. They also conducted a longitudinal *in situ* study, collecting four months of data. Moreover, they also targeted industry practitioners since the majority of their participants reported proprietary affiliations. Kersten and Murphy distributed a call for volunteers among industry programmers, and 97 initiated the study. In the end, 16 participants fulfilled all ac-

tivity requirements and were considered subjects in the study. Based on these 16 participants, they collected both quantitative and qualitative evidence that MyLyn can make programmers more productive. The main difference between the *in situ* study by Kersten and Murphy and our ImpRec evaluations is their explicit focus on programmers and source code navigation. In contrast, the evaluation presented in this paper involves participants of other roles, and navigation of non-code artifacts.

3 Industrial Context Description

This section contains a general description of the case company and presents the CIA work task. Furthermore, this section discusses the challenges related to CIA identified during our initial interviews in the case company.

3.1 General Context

The studied development organization is part of a large multinational company in the power and automation sector. We further describe the context structured in six different context facets, according to the guidelines provided by Petersen and Wohlin [258]: 1) products, 2) processes, 3) practices, tools, and techniques, 4) people, 5) organization, and 6) market.

Products. The products under development constitute an automation system that has evolved for decades. The system is safety-critical (governed by IEC 61511 [238]) and, once deployed, is expected to run without interruption for months or even years. Five major system versions exist, each introducing many new features via minor update releases. The code base contains over a million lines of code, dominated by C/C++ and some extensions in C#. The automation system contains both embedded systems and desktop applications, e.g., an IDE for developing control programs according to IEC 61131-3 [240]. The system is SIL2 safety-certified, according to IEC 61508 [239].

Processes. Projects follow a stage-gate iterative development process that is tailored to support the safety certification activities performed prior to release [121]. Prioritized features are added incrementally, followed by extensive testing. The most critical parts of the system are developed as redundant components by non-communicating teams at different development sites. All parts of the system are documented in detail, and the documents map to the (vertical) abstraction levels of the V-model e.g., system requirements, product requirements, functional requirements, detailed design specifications, and the corresponding artifacts on the testing side of the V-model.

Practices, tools, and techniques. The case company applies several established software engineering practices related to software quality. All code is reviewed both at commit-time and in formal review meetings. Documents are also reviewed in formal meetings. Several static code analysis tools run nightly on the

source code and several unit test suites run daily as automated tests, and code coverage is measured. Testing on the product and system level is conducted by an independent testing organization, but communication with developers is encouraged.

People. Hundreds of engineers work in this context. The company has a history that stretches more than a century, and the organization has a mature mix of experiences and ethnicities. Most engineers are male, but the genders are more balanced among junior employees. The roles directly involved in this investigation include developers, team leaders, managers, a safety engineer, and a configuration manager (cf. Table 2).

Organization. The studied organization is globally distributed. The main development sites are located in Europe, Asia and North America. Each development site is organized as a matrix structure, with development teams organized to satisfy project needs and a line organization offering various competences. The organization is strict, i.e., engineers rarely transfer between teams during projects.

Market. The product is released to a small market with only few players. Important customers in various market segment, e.g., oil & gas, or pulp & paper, sometimes initiated by bespoke development with specific feature requests. The market strategy is to offer the best possible automation system for very large industrial plants in the domain of process automation.

3.2 Change Impact Analysis in Context

CIA is a fundamental activity in safety-critical change and issue management. In the case company, all changes to source code need to be analyzed prior to implementation. The set of CIA analyses is a crucial component of the safety case, a documented argument providing a compelling, comprehensive, and valid case that a system is acceptably safe for a given application in a given operating environment [268]. Providing the external safety assessor with a high-quality safety case is among the top priorities of the organization under study.

The safety engineers at the case company have developed a semi-structured CIA report template (cf. Table 1) to support the safety case in relation to the IEC 61508 safety certification [239]. Developers use this template to document their CIA before committing source code changes. Six out of thirteen questions in the CIA template explicitly ask for trace links, see questions 4, 5, 6, 7, 8, and 12 in Table 1.

In addition to documenting source code changes, IEC 61508 mandates that the developers must also specify and update related development artifacts to reflect the changes, e.g., requirement specifications, design documentation, test case descriptions, test scripts and user manuals. Furthermore, the CIA report should specify which high-level system requirements are involved in the change, and which test cases should be executed to verify that the changes are correct once implemented in the system.

Table 1: The CIA template used in the case company, originally presented by Klevin [279]. Questions in bold font require the developer to specify explicit trace links.

Change Impact Analysis Questions	
1	Is the reported problem safety critical?
2	In which versions/revisions does this problem exist?
3	How are general system functions and properties affected by the change?
4	**List modified code files/modules and their SIL classifications, and/or affected safety safety related hardware modules.**
5	**Which library items are affected by the change?** (e.g., library types, firmware functions, HW types, HW libraries)
6	**Which documents need to be modified?** (e.g., product requirements specifications, architecture, functional requirements specifications, design descriptions, schematics, functional test descriptions, design test descriptions)
7	**Which test cases need to be executed?** (e.g., design tests, functional tests, sequence tests, environmental/EMC tests, FPGA simulations)
8	**Which user documents, including online help, need to be modified?**
9	How long will it take to correct the problem, and verify the correction?
10	What is the root cause of this problem?
11	How could this problem been avoided?
12	**Which requirements and functions need to be retested by the product test/system test organization?**

Developers conduct CIAs regularly during development and maintenance projects, and the CIA activity requires much effort. The developers report that they on average conduct CIAs weekly or bi-weekly during projects. The average time they spend on a CIA report depends on both the component of the system and the experience of the developer, ranging from fifteen minutes to several hours.

Several challenges are involved in the CIA process. In initial interviews to define the scope of the study, developers and safety engineers at the case company confirm that it is difficult to identify how a change affects the software system under development. Example challenges highlighted by the developers include: 1) side effects and ripple effects, 2) identifying impact on the system requirements, and 3) analyzing impact on quality attributes such as performance. Developers also reported organizational challenges such as: 4) building enough confidence in their own CIA report, and 5) recognizing the value of the comprehensive CIA process. Some developers report a slight aversion toward the CIA, which is considered a mundane activity that requires extensive browsing of technical documentation.

Currently there is little tool support available for CIA in the organization. The CIA process is tightly connected with the issue management process, as all changes to formal development artifacts require an issue report in the issue repository. All completed CIA reports are stored in the issue repository as attachments to issue reports, but there are few features available apart from general database functionality. Developers typically access the issue repository using a simple web interface. Only rudimentary browsing and searching is supported in the issue repository, e.g., filtering issues using basic metadata and searching using keywords. There is no support for full-text search, and the CIA reports (as they are attachments) are not indexed by any search engine at all. The only tool related to CIA is *IA Sidekick*, an internally developed tool, supports only input to the CIA template by providing formatting and basic sanity checks.

4 Approach and ImpRec

This section presents an overview of our solution proposal, as well as a brief description of the implementation of our ideas in the tool ImpRec [75].

4.1 Traceability Reuse for Impact Analysis

Developers at the case company put much effort into producing high-quality CIA reports, as described in Section 3.2. However, the content of the CIA reports is mainly used to create a strong safety case for certification purposes. A typical developer authors CIA reports to comply with the safety process, but does not consider them again once accepted and stored in the issue repository.

One of the challenges related to the CIA identified in the case under study (see Section 3.2) is that developers do not acknowledge the value of the rigorous CIA

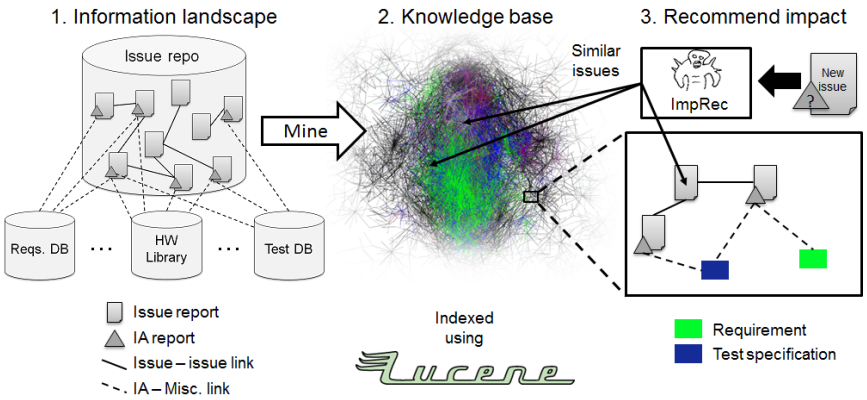


Figure 1: Overview of the approach implemented in ImpRec. First, the issue repository is mined to establish a knowledge base. Second, ImpRec identifies similar issue reports in the knowledge base using Apache Lucene and recommends impact based on the network structure.

process, i.e., it is regarded as an activity conducted solely to satisfy external stakeholders. Our approach addresses this negative perception by enabling developers to reuse knowledge from previously completed CIA reports [71]. By extracting trace links from the semi-structured CIA reports to previously impacted artifacts (a.k.a. link mining [195]), we establish a *knowledge base* of historical impact.

Once the knowledge base has been established, ImpRec uses it to recommend potentially impacted artifacts for new incoming issue reports. As such, ImpRec lets developers reuse the collaboratively constructed trace link network, i.e., "to follow in the footsteps of previous developers in the information landscape". Figure 1 shows an overview of our approach. The two main steps, mining the knowledge base and recommending impact, are described in Section 4.2 and Section 4.3, respectively.

4.2 Mining Previous Impact

As a first step in our approach, we construct a knowledge base of previous CIAs by conducting link mining in the issue repository (cf. the horizontal arrow in Fig. 1). First, we extract the "related issue" links between issue reports, stored in an explicit field in the issue tracker [73]. Then, we use regular expressions to extract trace links from the issue reports with attached CIA reports. As developers in the case company must use formal artifact IDs to report impact, regular expressions can capture all correctly formatted trace links.

Trace links in a CIA report are structured by the CIA template (cf. Table 1) and thus belong to a specific question. Consequently, we can deduce the meaning

of most of the extracted trace links, e.g., answers to Q7 and Q12 are related to verification. Another heuristic we rely on is that requirements and HW descriptions IDs have distinguished formats. Finally, we store all extracted traces (i.e., triplets of source artifact, target artifact, and trace link [203]) in a knowledge base represented by a semantic network [435].

The knowledge base contains 32,000+ issue reports and 13,000+ CIA reports from about a decade of software evolution. During this time period, developers have pointed out almost 2,000 unique non-code artifacts as impacted, categorized as requirements, test specifications, hardware descriptions, or miscellaneous artifacts (when no type could be deduced). Moreover, the knowledge base contains trace links of the following types (and count): specified-by (5,000+), verified-by (4,000+), needs-update (1,500+), impacts-HW (1,500+), and trace links whose type could not be determined (1,000+). Finally, the knowledge base contains 22,000+ related-to links between issue reports. In the visual representation in Figure 1, some of the semantic type information is encoded using colors.

4.3 Recommending Potential Impact

When the knowledge base is established, recommendations are calculated in three steps as described in Borg and Runeson [75]: 1) identification of textually similar issue reports (cf. Fig. 1), 2) breadth-first searches to identify candidate impact, and 3) ranking the candidate impact.

First, we use IR techniques to identify similar reports, referred to as *starting points* in the knowledge base. Apache Lucene, a state-of-the-art OSS search engine library [215], is used for the similarity calculation. Both terms in the title and description of issue reports are considered, after stemming and stop word removal. The first step in the recommendation process is in line with previous work on duplicate detection of issue reports [78], IR-based trace capture [31], and content-based RSSEs [179].

Second, we perform breadth-first searches in the knowledge base to identify candidate impact. Artifacts reported as impacted by the starting point i are added to a set of potentially impacted artifacts, the *impact set* (SET_i). Then, related-to links are iteratively followed from the starting points to extend the impact sets. This second step is inspired by collaborative RSSEs [179], and attempts to help the user to follow in the footprints (traces) of previous developers.

Third, we rank the artifacts in the impact sets. As multiple starting points are identified, the same artifact might appear in several impact sets. Thus, the final ranking value of an individual artifact (ART_x) is calculated by summarizing the contributions to the ranking value for all impact sets containing ART_x:

$$Weight(ART_x) = \sum_{ART_x \in SET_i} \frac{ALPHA * CENT_x + (1 - ALPHA) * SIM_x}{1 + LEVEL * PENALTY} \quad (1)$$

where SIM_x is the similarity of the issue report that was used as starting point when identifying ART_x, $LEVEL$ is the number of related issue links followed from the starting point to identify ART_x, and $CENT_x$ is the centrality measure of ART_x in the knowledge base. $ALPHA$ and $PENALTY$ are constants that enable tuning for context-specific improvements [70].

4.4 ImpRec: An RSSE for Change Impact Analysis

We implemented our approach in the prototype tool ImpRec[2], an RSSE for CIA. To ease deployment and to lower the training effort of the developers, we developed ImpRec as a .Net extension to IA Sidekick, an existing suite of CIA support tools. ImpRec evolved in close collaboration with developers in the case company, and our development effort was guided by continuous feedback.

ImpRec was directly developed for industrial-scale software development. We let the large number of artifacts work for us, as ImpRec leverages on a large knowledge base. Also, ImpRec scales well, as both the semantic network representing the knowledge base, as well as the search engine index provided by Apache Lucene can manage a large amount of information without performance issues.

Figure 2 shows the ImpRec GUI. The input area, denoted by A, is the first area the user interacts with. The user can paste the title and/or the description of the current issue report to trigger recommendations, the user can also conduct general free-text searches. The lower parts of the ImpRec GUI are used to present ranked recommendations. B shows a list view with similar issue reports in the knowledge base, and E lists potentially impacted artifacts.

ImpRec also implements a feedback mechanism, developed to support evaluation of the collected data (further described in Section 5.4). All items in the list views B and E have check boxes used by to denote that the corresponding recommendation is relevant for the ongoing CIA. Every time a developer starts ImpRec, a unique session is created. During the session, the following user actions are recorded:

- Search - The developer clicks the 'Search' button, next to A. All information related to the query is stored.
- Selection - The developer selects an item in the list view B or E.
- Relevant - The developer toggles the check box of an item in list view B or E.
- CancelRelevant - The developer untoggles a check box in the list view B or E.

[2]The name ImpRec refers to an imp, a mischievous little creature in Germanic folklore, always in search of human attention. Imps could also be helpful however, and Shivaji *et al.* recently envisioned imps sitting on the shoulders of software developers to guide them [404]. The authors also implemented a virtual imp, using machine learning, to enable prediction of software changes that introduce defects.

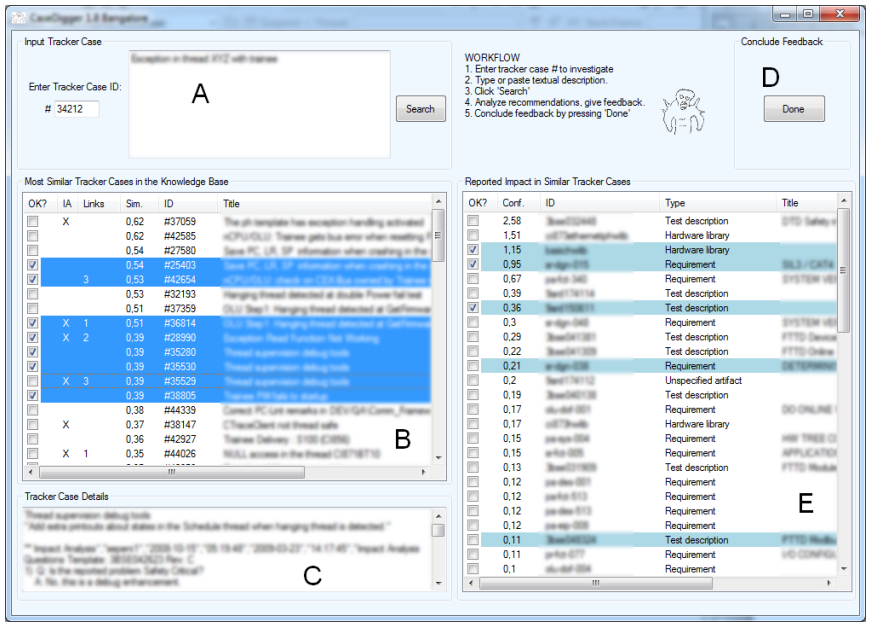


Figure 2: The ImpRec GUI. A: The search field. B. List of similar issue reports. C. Detailed information view. D. Feedback button. E. List of recommended impact. Selected parts purposely blurred.

- Confirmation - The developer clicks the 'Done' button (D), to conclude the feedback of the ongoing CIA task.

The user interface of ImpRec was designed according to the five factors that RSSE developers must consider according to Murphy-Hill and Murphy [346]: 1) understandability, 2) transparency, 3) assessability, 4) trust, and 5) distraction. While ImpRec still evolves, below we discuss some initial GUI decisions based on these factors.

Distraction does not apply to ImpRec, as the users initiate searches on their own. The *understandability* of the ImpRec recommendations is supported by the users' experiences of general search tools. Thus, also the *assessability* is supported; developers are used to assess items with a relevance that is predicted to decrease further down on a ranked list. Still, the separation of search results in two ranked lists (B and E) might not be obvious, and thus thoroughly explained in the user manual, available on the companion website[3]. Moreover, to further support assessibility, when a user clicks on an item in B, the full description of the issue report is presented in D, complemented by any stored CIA reports.

The two most critical factors for the delivery of ImpRec's recommendations are *transparency* and *trust*. We assume that user trust can only be built from a history of correct recommendations, and thus we focus on transparency. We increase transparency in two ways. First, the output of the ranking functions is presented to the user (i.e., the Apache Lucene similarity score in B, and the result of the ImpRec ranking function in E). This decision is in contrast to general search tools, but it might help expert users to understand the value of the recommendations. Second, when the user selects issue reports in B, the items in E that were reported in the corresponding CIA report are highlighted. Items that frequently were reported as impacted obtain a high ranking, and the user can observe this phenomenon while browsing the GUI.

5 Research Method

In this section we outline the research steps of this study and summarize the undertaken research method. Figure 3 provides an overview of the study.

Rationale and Purpose. This work was triggered by articulated issues at the case company associated with CIA for safety-critical development, and the potential solutions identified in the surveyed literature [76]. CIA is a fundamental part of the development and maintenance processes, and a requirement for safety certification of computer controlled systems. CIA is particularly challenging in the studied context due to a need for determining impact on non-code artifacts. Therefore, this research was conducted with an aim to support the developers by increasing the level of CIA automation using ImpRec.

[3] http://serg.cs.lth.se/research/experiment-packages/imprec/

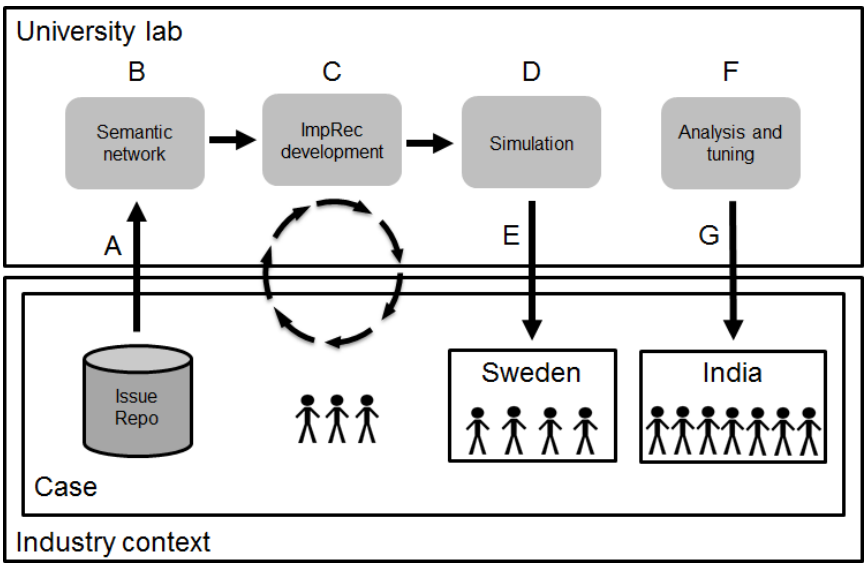


Figure 3: Overview of study design. A: Data extraction from issue repository. B: Knowledge base established in the form of a semantic network. C: ImpRec iteratively developed with short feedback loops. D: Static validation based on simulation. E: ImpRec deployed in Team Sweden. F: Intermediate results analyzed and tuning of ImpRec [70]. G: Tuned ImpRec deployed in Team India.

Table 2: Study participants in the dynamic validation.

Unit	ID	Team	Role	Degree year	System exp.	Classification
	A	Safety team	Safety engineer	1999	1999 ->	Senior, seasoned
	B	R&D	Development manager	2001	2001 ->	Senior, seasoned
	C	Protocols	Technical manager	2004	2005 ->	Senior, seasoned
Sweden	D	I/O	Team leader	2002	2004 ->	Senior, seasoned
	E	I/O	Developer	2007	2008 ->	Senior, seasoned
	F	I/O	Developer	1995	2014 ->	Senior, newcomer
	G	I/O	Developer/CM	2012	2012 ->	Junior, newcomer
India	H	Protocols	Team leader	2004	2005 ->	Senior, seasoned
	I	Protocols	Developer	2004	2010 ->	Senior, seasoned
	J	Protocols	Developer	2011	2011 ->	Junior, newcomer
	K	Protocols	Developer	2013	2013 ->	Junior, newcomer
	L	Protocols	Developer	2007	2007 ->	Senior, seasoned
	M	Protocols	Developer	2001	2007 ->	Senior, seasoned
	N	Protocols	Product manager	1994	2005 ->	Senior, seasoned

The case and units of analysis. The investigated case is *the formal CIA work task at the case company.* Two units of analysis were investigated in this study, named Unit Sweden and Unit India, see Figure 3. In both units of analysis, our aim was to investigate how using ImpRec can support engineers conducting formal CIA. All study participants are listed in Table 2. Developers in Unit Sweden and Unit India and three senior engineers (two from Sweden and one from India) were involved in the study.

Unit Sweden consisted of four developers in Sweden from the I/O team. We distributed a call for participation among all 60 developers at a site in Sweden. Three developers volunteered, and they happened to all be members of the same team working on safety-critical I/O communication. The three developers all had different responsibilities and experiences, offering the variety of perspectives suitable for a case study. To include yet another perspective in the study, we also asked the newest member of the team to join. As a sanity check, we examined the project history and found that the CIA frequency of the I/O team was close to the average among the teams at the development site.

The second unit of analysis included seven developers in India from the Protocols team, working with the same issue repository as Unit Sweden. The seven developers were selected to cover as many perspectives as possible, and all of them accepted to join the study.

Overview of the study. The study comprised three main phases: incremental ImpRec *development* in collaboration with the company (A-C in Fig. 3), *static validation* using simulation (D), and *dynamic validation* (E-G) as recommended by Gorschek *et al.* [201]. The first step of the development involved extracting all history from the issue repository at the case company (A) and mining the unstructured CIA reports to create a semantic network of software artifacts (B), further described in Section 5.2. Based on the semantic network, we iteratively developed

ImpRec as described in Section 4. The static validation was conducted using *in silico* simulation, in which ImpRec was applied to the historical inflow of issue reports (D), as proposed by Walker and Holmes [466]. Promising results from the static validation lead us to deploy ImpRec in Unit Sweden (E), i.e., we initiated dynamical evaluation through a longitudinal *in situ* case study as defined by Runeson *et al.* [413]. We used the initial results to tune the parameter settings of ImpRec (F) before deploying the RSSE in Unit India (G). The dynamic validation is further described in Section 5.3.

The steps involved in the study represent several years of research. The data collection (A) was conducted in the end of 2011. The mining involved in establishing a semantic network (B) was conducted and improved during 2012, and resulted in two publications [71, 73]. ImpRec was continuously developed and improved from 2013 [75], up to date. The static validation (D) was performed in the end of 2013, but the simulations were repeatedly executed as regression tests during ImpRec evolution. We deployed ImpRec in Unit Sweden in March 2014 (E), did tuning during the following Summer (F), and deployed the RSSE in Unit India (G) in August 2014. We conducted post-study interviews and concluded data collection in December 2014.

5.1 Research Questions

The following research questions guide our study:

RQ1 How accurate is ImpRec in recommending impact for incoming issue reports?

RQ2 Do developers consider the level of accuracy delivered by ImpRec sufficient to help when conducting change impact analysis?

RQ3 Do newcomer developers benefit more from ImpRec than developers that know the system well?

We addressed RQ1 using static validation in the form of computer simulations, see Section 5.2. By applying established IR measures, i.e., precision and recall, we enable comparisons with previous work, but we also go beyond set-based measures by reporting Mean Average Precision (MAP) [325].

We tackled RQ2 and RQ3 by deploying ImpRec in two development teams. RQ2 deals with mapping the quantitative results from RQ1 to actual user satisfaction. How accurate must an RSSE be before developers start recognizing its value? Has ImpRec passed the utility breakpoint, as discussed by Regnell *et al.* in the QUPER model [390]?

RQ3 is an attempt to corroborate an assumption from previous research that newcomer developers benefit the most from RSSEs. Čubranić developed Hipikat particularly to support developers new to OSS projects [126], and ten years later Panichella also developed RSSEs for OSS project newcomers [366]. We aim to

find supporting, or contradicting, empirical support that newcomers benefit more from ImpRec than seasoned developers.

5.2 Static Validation: *In silico* Evaluation

Our approach to initially justify our RSSE is to conduct a *simulation* [466]. To minimize potential confounding factors, this step was conducted without human subjects by comparing recommendations from ImpRec with the results from the historical CIA reports. This static validation assesses the *correctness* of ImpRec as defined by Avazpour *et al.* [32] , i.e., "*how close the recommendations are to a set of recommendations that are assumed to be correct*". Since the historical CIA reports have undergone a thorough review process, due to their key role in the safety case, it is safe to assume that they are correct.

Simulations are performed by scientists and engineers to "*better understand the world when it cannot be directly studied due to complexity, costs, or risks*" [466]. By imitating the environment where ImpRec will be deployed, i.e., the inflow of issue reports at the case company, we could examine the correctness of the recommendations. We collected 26,120 chronologically ordered issue reports and 4,845 CIA reports from the last 12 years of development. We divided these data into a training set (88%) and a test set (12%). To ensure a realistic simulation, we do not filter the dataset in any way.

We established a knowledge base from the training set of (all) 4,249 CIA reports submitted prior to July 2010. The test set contained issue reports submitted between July 2010 and January 2012. Among the issue reports in the test set, there were 596 CIA reports. In the simulation, we used the titles of the associated issue reports as queries to ImpRec. We considered the 320 trace links from these CIA reports to various non-code artifacts as our gold standard. Among these 320 trace links, 20% of their target artifacts had not been reported as impacted during the 10 years of evolution represented in the knowledge base. Consequently, as ImpRec relies on developers' previous work, the catalog coverage [32] in the simulation was 80%, i.e., only 80% of the artifacts we wanted to link were available in the knowledge base (cf. the horizontal 'ceiling' line in subplot a) Fig. 4).

We compare two configurations of ImpRec against two baselines representing naïve strategies. ImpRec A (deployed in Unit Sweden) and ImpRec B (deployed in Unit India) constitute the two configurations deployed, before and after systematic parameter tuning (presented in detail in previous work [103]). ImpRec Text is a baseline that only relies on textual similarity, simply returning the artifacts previously reported as impacted in the 20 most similar issue reports in the issue repository, ranked by the number of occurrences. ImpRec Cent is a baseline that only uses the network structure in the knowledge base. The baseline always reports the artifacts with the highest centrality measures, no matter the textual content of the incoming issue report.

5.3 Dynamic Validation: *In situ* Evaluation

While the results from an *in silico* evaluation can indicate the usefulness of an RSSE, user studies must be conducted to gain deeper understanding [402]. To study ImpRec in the full complexity of an industrial context, we performed a longitudinal *in situ* evaluation. By letting the developers in Unit Sweden and Unit India use ImpRec during their daily work, we complemented the assessment of correctness (RQ1) with *utility*, i.e., the value that the developers gain from the recommendations (RQ2).

The participants first received ImpRec instructions and a research study description in an initial face-to-face meeting. The instructions clarified the purpose of the study, and how the participants should use ImpRec in their CIA process. During the meeting we also presented a demo of ImpRec and distributed the user manual.

Before deploying ImpRec, we conducted semi-structured interviews with all participants individually, and three additional senior engineers (A-C in Table 2). The interviews[4], roughly 45 min long, covered the state-of-practice CIA work task, and challenges experienced by the participants. Also, we discussed some specific CIA reports authored by the interviewee, as well as two measures extracted from their recent CIA history (10-30 CIA reports per interviewee):

i) the time from the assignment of an issue report to a developer until a CIA report is submitted (*TimeCIA*: reflecting the effort required per CIA)

ii) the number of modifications to an CIA report after submission (*ModCIA*: indicating the quality of the first CIA).

We concluded the interviews by installing ImpRec on the participants' computers followed by a brief tutorial. We also clarified that all actions performed in ImpRec would be stored in a local log file, as described in Section 4.4. After the interviews and the demo, all participants felt that they were ready to use ImpRec in their daily work. We offered them the possibility to email their questions or get additional training if needed. We also instructed all participants that our study was longitudinal, planned for several months, and that reminders would be sent regularly during the study.

Throughout the study we received partial log files through email, complemented by qualitative feedback, a type of communication we strongly encouraged. We collected final versions of the user log files in December 2014.

After concluding the data collection, we conducted post-study interviews with the participants in Unit Sweden, about 30 min each, also following an interview guide. Feedback and reflections from Unit India were collected via email. The main goal of the post-study interviews was to discuss utility focused on quality

[4]The interview guide is available on the accompanying web site [78]. The full analysis of the interviews will be reported in a separate publication.

breakpoints as introduced in the QUPER model [388]. The QUPER model considers quality to be an inherent characteristic on a sliding scale, but non-linear in its nature. We asked the interviewees to assess what the correctness of ImpRec represents, as well the correctness of the current manual CIA approach, compared to the three breakpoints of QUPER:

1. *Utility*: the user starts recognizing the value of the tool. A low quality level, and anything below is useless.

2. *Differentiation*: (a.k.a. "wow!") the tool starts to become impressive, and users definitely would use it if available.

3. *Saturation*: (a.k.a. "overkill") increased quality beyond this point is not of practical significance.

Huffman Hayes *et al.* have proposed similar evaluation ideas for mapping quantitative output from software engineering IR tools to quality levels [174]. However, while they presented initial threshold based on their own experiences, we try to explore quality levels based on interviews with developers. The goal of our utility assessment is primarily to determine whether the accuracy of the recommendations provided by ImpRec has passed the utility breakpoint (RQ2), i.e., "the point where developers start to recognize the value of the tool" [293].

5.4 Measures and Analysis

In the *in silico* static validation, we quantified the *correctness* based on the fraction of the gold standard recommended by ImpRec. We define a *true recommendation* as a suggestion from ImpRec that is also present in the corresponding CIA report, and a *useful recommendation* as either true or explicitly confirmed as relevant by a participant. We report set-based measures rather than averages per query (a.k.a. matrix-based IR evaluation [76], or micro-evolution [497]). *Recall* is the fraction of the true impact that ImpRec recommends (max 80% in the simulation). *Precision* is the fraction of the ImpRec recommendations that indeed represent true impact. *F1-score* is the harmonic mean of precision and recall, without favoring one or the other. However, as several researchers argued that recall is more important than precision in tracing experiments [118, 144, 231], we also report *F2-score, F5-score*, and *F10-score*, corresponding to a user who attaches 2, 5, and 10 times as much importance to recall as precision [460]. *Mean Average Precision* (MAP) is a secondary IR measure [77], taking also the ranking of retrieved items into account.

As argued by Spärck Jones *et al.*, pioneers of IR evaluation, only reporting precision at standard recall levels is opaque [438]. The figures obscure the actual number of recommendations needed to get beyond low recall. Thus, we report IR

measures for different cut-off points, representing between 1 and 50 recommendations from ImpRec. Showing only one recommendation per CIA would not be very useful, and recommending too many also brings no value.

To assess the *utility* in the *in situ* dynamic validation, we performed triangulation of data from semi-structured interviews and collected log files. The interviews before and after deploying ImpRec were recorded, transcribed word-by-word, and sent back to the interviewees for validation within two weeks after the interview. We conducted thematic analysis [123] of the initial interviews, and for the post-study interviews we sought qualitative comments related to the findings in the user log files. Further qualitative feedback, informal but highly valuable, was collected in meetings and e-mails during the study.

The log files collected from the users contain rich information, and thus enable Search Log Analysis (SLA) [244]. We primarily used the search logs to study the explicit feedback functionality (see Section 4.4), but we also compared the recommendations against the final CIA reports stored in the issue tracker (when available).

Finally, we studied how the developers interacted with the ranked recommendations, and how much time they spent browsing the results. We report the *click distribution*, i.e., the frequency of clicks distributed across different positions on the ranked list, however only for related issues (B in Fig. 2) as there is no incitement for the participants to click on individual items among the potential impact (C in Fig. 2).

6 Results and Interpretation

This section presents the results from our evaluation, as well as the corresponding interpretation.

6.1 Static Validation: *In silico* Evaluation

Figures 4 and 5 portray the correctness of ImpRec's recommendations from the *in silico* simulation. The graphs show precision, recall, MAP, and F-scores from a simulation, using an unfiltered set of issue reports, constituting 10 years of software evolution at the case company, as described in Section 5.2.

The four subplots a)-d) in Figures 4 and 5 follow the same structure. The y-axes show the different IR measures, all with values between 0 and 1. The x-axes depict the cut-off point of the ranked list, N, i.e., how many ImpRec recommendations are considered. The subplots display four different ImpRec configurations: A (solid line), B (dashed line), Text (dot-dashed line), and Cent (dotted line). As presented in Section 5.2, Text and Cent constitute two naïve baselines relying fully on textual similarity and centrality measures, respectively.

Figure 4 a) presents how the recall improves as N increases. ImpRec A performs the best (among the four configurations) before N exceeds 10 (Rc@5=0.33

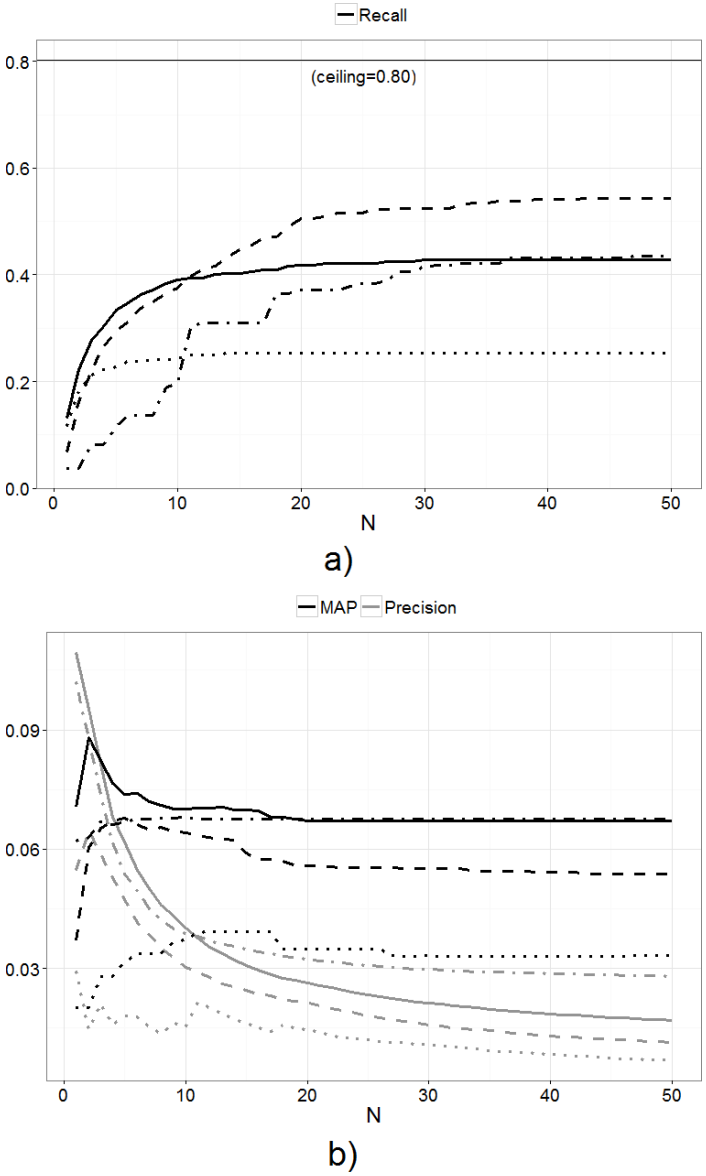


Figure 4: Recall, precision, and MAP from the static validation, i.e., the *in silico* simulation. Solid line: ImpRec A, dashed line: ImpRec B, dot-dashed line: ImpRec Text, and dotted line: ImpRec Cent.

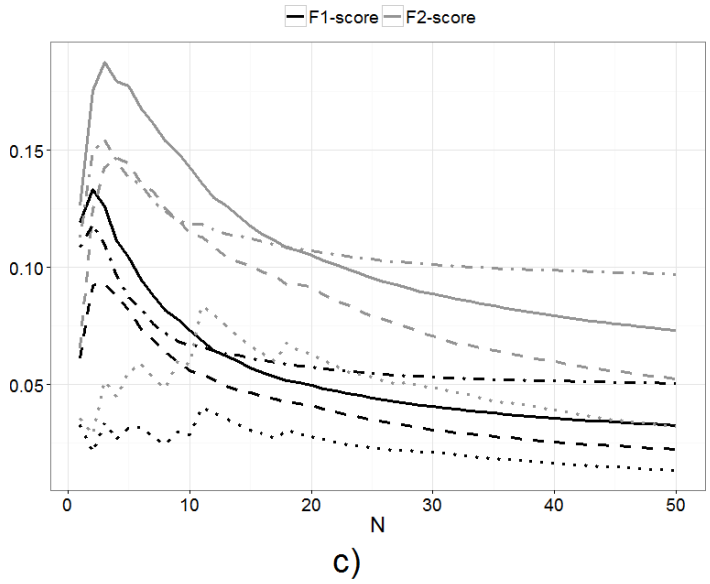


Figure 5: F-scores from the static validation, i.e., the *in silico* simulation. Solid line: ImpRec A, dashed line: ImpRec B, dot-dashed line: ImpRec Text, and dotted line: ImpRec Cent.

and Rc@10=0.39), but there is little improvement as the number of recommendations increases further. ImpRec B performs slightly worse than ImpRec A for N<10, but improves steadily until Rc@20=0.51. Regarding the two baselines, we conclude that ImpRec Text (dot-dashed line) is imprecise for few recommendations, but for large N, the recall matches the level of ImpRec A. The low initial recall (Rc@1-9<0.20) of the purely textual baseline shows that it does not capture all variations of the natural language. However, it appears that the naïve textual approach matches recall levels of a more advanced configuration for N>30, i.e., if the user accepts sifting through a high number of recommendations. ImpRec Cent (dotted line) on the other hand displays a notably worse recall curve.

Figure 4 b) shows how precision drops as N increases (gray lines). ImpRec A, B, and Text show similar declines, with ImpRec Text being somewhat better at N>10. Our results show that while a purely textual approach to recommending impact does not cover everything in the gold standard at low N (Rc@5<0.1), the textual similarity appears to indicate helpful historical issue reports with a reasonable precision (Pr@5>0.06). Again ImpRec Cent results in the worst recommendations (Pr <0.03 for all N). Figure 4 b) also shows how MAP changes with increasing N (black lines). Among the four configurations, ImpRec A shows the best MAP at N<19, at larger N ImpRec Text performs equivalently. ImpRec B and ImpRec Cent generally perform worse wrt. MAP (e.g., Map@20=0.057 and MAP@20=0.032, respectively).

Figure 5 c) shows F1-scores (black lines) and F2-scores (gray lines), i.e., a combined measure treating recall equally important, or twice as important, as precision. When the number of recommendations is low (N<10), ImpRec A displays the best F1- and F2-scores. The highest F1- and F2-scores correspond to two or three recommendations from ImpRec A, respectively.

Figure 5 d) presents F5-scores (gray lines) and F10-scores (black lines), i.e., evaluation measures greatly emphasizing recall. The highest F5- and F10-scores correspond to nine recommendations from ImpRec A and twenty recommendations from ImpRec B, respectively.

The *in silico* simulation suggests that reusing traceability established by previous developers is a feasible approach to support non-code CIA. As reported in Section 5.2, ImpRec's catalog coverage in the evaluation is 80%, i.e., a majority of the non-code artifacts impacted by issue reports in the test set had been reported as impacted before. We also show that the ImpRec ranking function appears to be useful, as roughly 30% of the true impact in the gold standard is recommended among the top-5 candidates, and 40% among the top-10 candidates. ImpRec B is the configuration that comes the closest to the upper limit, with Rc@50=0.54, i.e., 54% of the true impact is recommended among the top-50 candidates.

Table 3 lists the correctness of ImpRec compared to the related work presented in Section 2.3. The figures can not be directly compared as: 1) ImpRec recommends non-code artifacts and the related work addresses code impact, and 2) the projects studied are different. Nevertheless, the results indicate that ImpRec per-

Table 3: ImpRec correctness compared to previous work.

	Study	Pr	Rc	Project(s)
@10	ImpRec	0.05	0.40	Automation
	Canfora and Cerulo [92]	0.20	0.40	Gedit
		0.05	0.20	ArgoUML
		0.15	0.90	Firefox
	Gethers *et al.* [194]	0.15	0.20	ArgoUML
		0.10	0.20	JabRef
		0.15	0.35	jEdit
		0.10	0.25	muCommander
	Zimmermann *et al.* [497]		0.33 (avg.)	Eclipse, gcc, Gimp, JBoss, jEdit, KOffice, Postgres, Python
@?	Čubranić *et al.* [127]	0.10	0.65	Eclipse
	Ying *et al.* [486]	0.5	0.25	Mozilla

forms in line with previous work on CIA. Focusing on other evaluations reporting the correctness of the top-10 recommendations, we observe that the precision of ImpRec is below average, but the recall is among the best. This result mirrors our aim to reach high recall within a reasonable (browsable) amount of recommendations.

To summarize, the static validation results in correctness, i.e., precision and recall measures, in line with previous work. We observe that the static validations of ImpRec, using a test set containing all issue reports submitted during 1.5 years of system development, reaches Rc@10=0.40. This result is competitive compared to the work presented in Section 2.3, only outperformed by the approach by Canfora and Cerulo when evaluated on the Firefox project. Regarding precision however, ImpRec appears to perform in the lower end. As we consider recall to be more important however, at least at reasonable levels of N, we consider the correctness of ImpRec to be good enough to initiate the dynamic validation.

6.2 Dynamic Validation: *In situ* Evaluation

This section describes the results from the dynamic validation, organized into: 1) overall results, 2) detailed results per participant, and 3) mapping correctness to utility.

Overview of the Results

The initial interviews confirmed the significance of the problem, and the considerable effort spent (already described in the context description in Section 3). The interviewees report that the frequency of the CIAs depends on the phase of the project and give two CIAs on average per month. However, the interviews refuted the value of the two measures TimeCIA and ModCIA defined in Section 5.3, as too confounded by other variables, and only useful as "a very rough indication of

effort and complexity" (participant A). Thus, we focus the dynamic validation on SLA combined with qualitative feedback from the post-interviews.

We received positive responses during initial installation and demonstration of ImpRec. Several participants expressed interest in trying the RSSE, and promised to provide further feedback. One participant immediately found helpful recommendations during the demonstration, saying "this [issue report] was exactly what I was looking for actually" (K). On the other hand, one participant (H) did not foresee any obvious use cases for the tool, indicating that its use might not be fully intuitive for all potential users.

In total, the participants conducted 43 ImpRec search sessions to explore CIAs related to issue reports, 33 times in Unit Sweden and 10 times in Unit India. The numbers reflect the different development phases and the extended period of data collection in Unit Sweden, see Section 5. Thirty-one of the search sessions concern issue reports that resulted in a completed CIA report during the study, i.e., we can directly compare them to the ImpRec output. In total 5 of the 43 uses did not result in an explicit 'confirmation' click by the user (cf. D in Fig. 2), but we could still perform partial analyses.

Table 4 shows descriptive statistics about the ImpRec sessions. First, five columns report general information about ImpRec usage: unit of analysis, participant ID, number of ImpRec sessions (in parenthesis: the number of related CIA reports stored at the end of the study), query-style of the user (T=copy/paste of title, D=copy/paste of title+description, U=user generated query, i.e., free text) incl. average number of characters/terms in the queries, and the average time per ImpRec session. The participants spent about five minutes per session with ImpRec. The Search Log Analysis (SLA) revealed that different search strategies were used. Six users used manually crafted queries as input to ImpRec (U in the fourth column), typically important keywords from the domain. These users used several different queries per session, but they were often restricted to a few terms. Participant D explained that he "started with broad searches, and then tried to gradually make them more specific". On the other hand, participant E exclusively used long queries (avg. 101 terms), consisting of both the title and the description of issue reports, stating that "I didn't think of any patterns really, but I think that's how you should search". Three users mostly used titles as search queries, on average containing ten terms.

Table 4 also shows the accuracy, per participant, of the ImpRec recommendations. Second, four columns show results concerning related issue reports: number of related issue reports in the gold standard (in parenthesis: number of related issue reports covered by the knowledge base), number of true related issue reports recommended by ImpRec, number of useful but not true recommendations, and the corresponding result in term of recall (in parenthesis: the maximum recall based on the knowledge base coverage). Finally, four columns show results regarding impacted artifacts, analogous to the related issues.

In total, the participants used the confirmation clicks to report that ImpRec pro-

Table 4: Detailed results, per participant, from the dynamic validation.

					Related issue reports				Impacted artifacts			
	ID	#Sessions	Query	Avg. time	#Gold	#True	#Extra	RcRel	#Gold	#True	#Extra	RcImp
Sweden	D	8 (4)	U: 13/66	2 min	20 (11)	2	10	0.1 (0.55)	24 (24)	5	4	0.21 (1)
	E	14 (14)	D: 101/501	9 min	24 (9)	6	5	0.25 (0.3)	62 (56)	28	7	0.45 (0.90)
	F	8 (4)	T: 9/47	8 min	10 (4)	2	13	0.2 (0.4)	33 (31)	14	7	0.42 (0.94)
	G	3 (3)	T: 10/49	7 min	7 (7)	0	2	0 (1)	32 (28)	1	1	0.03 (0.88)
India	H	3 (1)	U: 5/30	1 min	5 (3)	0	1	0 (0.6)	4 (1)	0	1	0 (0.25)
	I	1 (0)	T: 11/63	3 min	16 (6)	0	2	0 (0.38)	N/A	N/A	0	N/A
	J	2 (1)	U: 2/15	2 min	4 (4)	0	5	0 (1)	2 (1)	0	1	0 (0.5)
	K	1 (1)	U: 4/25	80 min	2 (0)	0	7	0 (0)	8 (0)	0	0	0 (0)
	L	Performed no change impact analyses during the study										
	M	1 (1)	U: 2/8	?	4 (4)	0	6	0 (1)	1 (1)	1	0	1 (1)
	N	2 (2)	U: 3/14	6 min	1 (1)	0	5	0 (1)	0 (0)	0	8	N/A
Sum		**43 (31)**										

vided relevant items in 30 of the 43 ImpRec sessions (70%). Useful related issue reports were provided for 49% of the sessions, and truly impacted artifacts in 56% of the sessions. We observe that for two participants that did several search sessions (E and F), the recall for impacted artifacts corresponds to the static validation (0.45 and 0.42).

The participants sometimes missed true recommendations provided by ImpRec. Among the 49 impacted artifacts that were true, participants missed 19 of them (39%). As expected, the position of recommendations on the ranked lists was important. The click distribution of recommended related issue reports (see Fig. 6) shows that the participants interacted more with recommended issue reports presented at the top of the list, and the decrease in clicks is similar to what has been reported for web search [227]. Participant G commented that "the first search hits felt good, but somewhere after 10 or 12 it turned wild", and participant E stated that "I don't think I looked beyond 10. The ranking function was quite good, I started trusting it".

ImpRec often provided relevant related issue reports that were not explicitly stored as 'related' in the issue tracker. In total, the participants reported that ImpRec presented 66 relevant related issue reports, and 56 of these relations (85%) were not explicitly stored in the issue tracker. This indicates that there is a large amount of issue reports that never are connected, despite that developers consider them related. Moreover, it suggests that the network of issue reports in the issue tracker, analyzed in previous work [73], underestimates the issue interrelations. Regarding potentially impacted artifacts, participants reported, by confirmation clicks, that ImpRec presented 78 truly impacted artifacts, and 29 of them (37%) were not reported in the formal CIA reports. This suggests that ImpRec can be used to complement manual work by recommending novel artifacts, thus improving the recall of CIA reports.

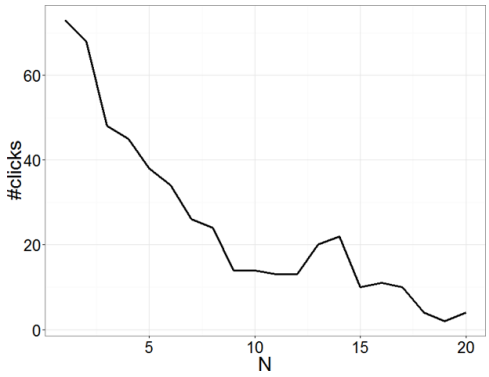


Figure 6: Click distribution of the participants' interaction with recommended related issue reports.

Detailed Results per Participant

Participant D, the team leader of Unit Sweden, conducted eight search sessions with ImpRec. Compared to the other participants in the study, his sessions were shorter, but conducted in an iterative fashion with short queries. ImpRec delivered relatively few true recommendations (RcRel=0.10, RcImp=0.21), but he confirmed several additional artifacts as relevant. Despite the rather poor quantitative results, he was positive about the approach and explained that a tool that reuses previous knowledge could help tracing changes to 'difficult' artifact types: "What we miss as developers are items we don't have a relation to. We know the code to change, which files and modules. But tracing to requirements, and the system tests as well, that's where a tool like this could help".

Participant E used ImpRec 14 times during the study, more than any other participant. He was also the only participant who mainly used full text descriptions as queries in ImpRec. Most of the issue reports related to his tasks were not available in the knowledge base, thus the RcRel of ImpRec was constrained to 0.3. Regarding RcImp (0.45) however, the results are in line with the static validation. During the post-study interview, (E) explained that he "already knew about most recommendations provided by ImpRec", and that only a few times ImpRec identified information that complemented his knowledge. This observation stresses that it is not enough to look at recall in isolation, as there is a risk that a high recall value contains nothing but obvious information. On the other hand, confirming the users' ideas is one of the goals of an RSSE [402] (the other goal is to provide novelty), and it is also critical in building the users' trust in the tool [32].

Participant F performed eight ImpRec search sessions, using titles of issue reports as queries. As for participant E, most related issue reports were more recent than what was covered by the knowledge base (RcRel constrained to 0.4).

On the other hand, ImpRec identified 13 meaningful issue reports that were not formally acknowledged in the issue tracker. Finally, RcImp (0.42) was in line with the static validation. Participant F expressed that he "absolutely found some of the recommendations useful", but also that he maybe did not get the most out of ImpRec as he "might not have used the tool properly" and that "any search hits that require scrolling to find might be missed, and if the first hits do not make sense, you stop looking". Thus, the post-interview with participant F confirmed that both proper tool instructions as well as an accurate ranking function are important.

Participant G used ImpRec three times during the study, even though he commented: "Did I do only three? It felt like at least seven". During the course of the study, participant G gradually shifted to a full-time Configuration Management (CM) role. This change decreased the number of issue reports assigned to him, and introduced CM related meta-issues, e.g., branching, for which ImpRec did not provide accurate recommendations. However, he explained that also when ImpRec did not provide directly related artifacts, it supported general system comprehension: "It was worthwhile to further investigate some recommendations, even though I didn't report them as relevant in the end. Doing so helped me understand how things go together". Still, ImpRec's poor results on meta-issues indicate that the approach is best suited for actual defect reports.

Participant H, the team leader of Unit India, conducted three ImpRec search sessions. However, only one of the sessions related to an issue report with a completed CIA report at the end of the study. She performed manually crafted queries and iteratively modified them. Only in one session she confirmed the results using the explicit feedback function, reporting two useful recommendations (one related issue report and one impacted artifact). Participant H assessed the search results very fast, on average in 1 minute. This behavior is partly explained by the iterative search strategy. We suspect that ImpRec might have delivered more useful recommendation if more effort was spent, but as explained by herself "as a team leader, I do not work directly with CIA reports as much as before".

Participant I used ImpRec only once during the study. The single issue report triggering the CIA had 16 related issues stored in the tracker, the highest number in the study. Still, only six of them were present in the knowledge base, and none of them were recommended (RcRel=0). On the other hand, ImpRec recommended two other issue reports that participant I confirmed as relevant. RcImp could not be calculated, as there was no available CIA report for the specific issue report.

Participant J conducted two search sessions during the study, and one of them had a corresponding CIA report. ImpRec did not provide her any true recommendations (RcRel=RcImp=0), but instead four useful related issues and one useful impacted artifact. The qualitative feedback confirmed the value of the recommendations. Participant J said that "Your tool helped me to get a list of all related issues. The issue that I was working on was raised in many earlier system versions, and different people apparently worked on that with no success". This statement again shows the importance of going beyond the simple recall measure when

evaluating an RSSE.

Participant K, the most junior developer in the study, used ImpRec only once. He used a short user generated query, and ImpRec recommended seven useful previous issue reports. However, neither the two relevant issue reports in the gold standard, nor the eight impacted artifacts, were covered by the knowledge base. Still, participant K was satisfied with his single ImpRec experience, explaining: "I used the tool for a crash issue I'm working on. I found it very useful as I was able to find some old issue reports with similar problems and how they were fixed". He confirmed results in ImpRec 80 min after the click on the search button, thus obviously doing other things in the meantime. We consider the time as an outlier, i.e., it is not used in the calculation of average time per ImpRec session.

Participant L was was the only participant who did not use the tool during the study. We consider this indicative of the individual variation in CIA frequency, in line with the assessments made by the participants during the initial interviews.

Participant M used ImpRec for one search session using a short two-word query. The query resulted in four useful (but no true) recommended related issues, and the only truly impacted artifact. He did unfortunately not store his full user log file, thus we could not perform a proper SLA.

Participant N represents the perspective of a product manager, a type of user that browses rather than writes CIA reports. He confirmed that ImpRec delivered several useful recommendations (five related issue reports and eight impacted artifacts) but none of them were in the gold standard. Participant N expressed worry, in line with participants F and M that he might not have used ImpRec properly: "I'm not sure how well I used it, I basically just looked at the CIA reports of the related issues". However, as this reflects a typical ImpRec use case, we do not share the participant's concern.

Mapping Correctness and Utility

The qualitative feedback from the developers enables us to map the quantitative results from the static validation, i.e., correctness measured in recall, to the experienced utility of ImpRec. We discussed the utility during post-study interviews with the participants in Unit Sweden, based on the QUPER model [390]. Figure 7 shows an overview of the discussions. Three developers found it useful to discuss quality and benefit based on the QUPER model. Respondent F instead preferred less structured discussion of utility, as he considered both the quality dimension and the subjective benefit as too abstract.

Participant E have conducted many CIA reports during his time at the company, and he explained that they are rarely modified once they have been submitted. "I do my best to answer the questions important to me, the document updates and the tests to run, but of course I try to answer the rest properly as well. Subjectively I would put my manual quality close to the saturation point". This statement reflects that higher recall in the CIA reports would have little practical signifi-

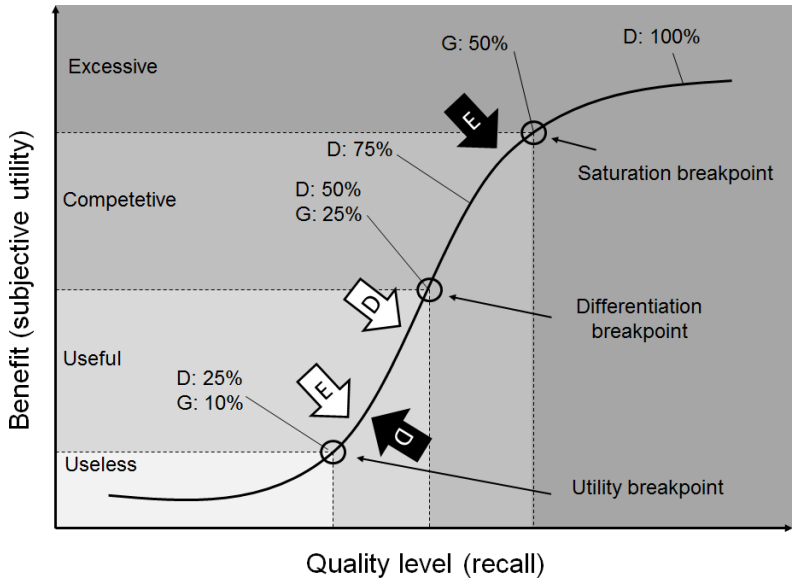


Figure 7: Mapping correctness and utility using the QUPER model. Black arrows depict the current manual work, white arrows represent working with ImpRec.

cance. Participant D put the current correctness of ImpRec in the lower end of the 'useful' region, and motivates his choice "I think I rarely got a really good search hit that I hadn't already thought of. Maybe just once or so. I don't think using the tool made me rethink the content of my CIA reports". This statement clearly shows that (E) did not receive much novelty from ImpRec, but rather confirmation. Participant E also quantified the QUPER quality breakpoints for ImpRec: utility - 25%, differentiation - 50%, saturation - between 75% and 100%.

Participant D focused more on the benefit dimension than the quality. His main consideration was that his experienced benefit of the CIA report was low with the current way of working, no matter how correct they would be: "During the project the value is very limited. We don't use the content of the CIA report properly, they just live on parallel to the project. Even if they would be 100% accurate, their benefit to me would still not be better than 'useful"'. However, he acknowledged that other roles than developers might find them more useful, especially at the end of projects. He also made a critical remark: "CIA reports is not a parameter in the project planning. But they should be." Regarding the utility of ImpRec, participant D said "Already this prototype is clearly useful". Moreover, he explained that the overall RSSE approach is promising: "What we developers are bad at is tracing to requirements. And reporting the test case selection back to the test organization. This is where I think there is potential in a tool like this, to take advantage of what

others have done before".

Participant G had a contrasting view on the quality levels. He preferred to view the quality as binary: "There are no different levels like this. CIA reports are either worthless or ok." He emphasized the variation of how skilful developers are at writing CIA reports, and that some tend to report too much. Thus, the interview revealed some of the politics involved in the safety process: "Some love to describe the whole world, but that typically hits you back. The safety team will polish the answers before sending them to [the (anonymized) external safety assessor]. Otherwise they will come back and demand re-implementation and testing, strangling the development for two years." Participant G explained that the organization aimed for "good enough" CIA reports, and then hoped that the overall process would ensure a decent level of quality. Finally, participant G stated that automated tool support for CIA could be useful, and quantified the QUPER breakpoints as follows: utility - 10% ("below that it would be too much waste of time"), differentiation - 25%, and saturation - 50% ("too much help is not good, I still want the developers to work for themselves").

Participant F was the only participant that preferred to discuss utility without the structure of the QUPER breakpoints. Instead, his impression was that the current manual CIA reports cover 75% of what should actually be reported. He acknowledged that ImpRec sometimes provided him with useful recommendations, but particularly commended the fast searches in the tool. He also encouraged future evolution of the RSSE as he believed in the approach of finding related CIA reports from the past. However, participant F also stressed other aspects of utility: "You become more inclined to use the tool if it is integrated in some way. Otherwise there is always this threshold. It has to be integrated and easy to use."

The RSSE integration aspects were also mentioned by participant D during the post-study interview. He requested ImpRec to be properly integrated in the issue tracker, to help the RSSE reach its potential. He also mentioned three other improvement proposals: 1) enabling filtering of the search results, 2) personalization of the searches, maybe by tagging 'favorites', and 3) introducing more templates, both for searching and writing answers to the CIA questions. Towards the end of the interview, participant D also warned about the dangers of tools like ImpRec, since also developers' erroneous decisions could be propagated to future CIAs.

We received positive qualitative feedback on the utility of ImpRec from Unit India as well, e.g., from participants K and M, however not structured according to the QUPER model. Some participants in Unit India were also eager to try ImpRec further, as represented by participant J asking "Could you tell me how to upgrade the database of the tool, so that it gets the latest set of issues?" The question puts the importance of keeping the knowledge base up-to-date in focus, an important direction for future work.

Summarizing the utility evaluation, we note that the developers' reception of ImpRec varies. Still, it appears that the level of correctness provided by ImpRec can support developers in the case under study. For example, all participants

in Unit Sweden shared some positive experiences from working with ImpRec. Participants D and E explicitly put ImpRec beyond the utility breakpoint in the QUPER model, and participant G associated the utility breakpoint with a quality level well below what ImpRec delivers (10%). Participant F preferred to discuss the utility in a more abstract fashion, but reported that ImpRec 'absolutely' presented some useful recommendations. The evaluation in Unit India, involving more participants (but only 23% of the total number of ImpRec search sessions), also indicated that ImpRec was helpful. The junior developers (participants J and K) confirmed usefulness of the tool, and two senior developers (participants M and N) reported several correct recommendations using the confirmation functionality. On the other hand, two other senior developers (participants H and I) did not provide any qualitative feedback, and also confirmed fewer recommendations as true.

7 Threats to Validity

We discuss threats to validity in terms of *construct validity*, *internal validity*, *external validity*, and *reliability* as proposed by Runeson *et al.* [413]. We minimize the conclusion validity discussion as the conclusions of this paper do not arrive from inferential statistics and its assumptions [477].

7.1 Static Validation (RQ1)

The main threats to our conclusions regarding correctness belong to construct validity (i.e., the relation between the theories behind the research and our observations) and external validity (i.e., whether the results can be generalized). As ImpRec is tailored for the specific context, we discuss generalizing the static validation to other sets of issue reports in the same organization, not to other companies.

All measurements of the ImpRec recommendations are made by comparing to the gold standard. Thus, our conclusions regarding the ImpRec's accuracy assume that the gold standard is correct. The gold standard is extracted from manually created CIA reports during years of software evolution, and it is likely that some of the reports point out to either too many impacted artifacts or too few. However, the CIA reports are among the most frequently reviewed artifacts in the organization, as they are fundamental to the safety certification. The change control board, project managers, and the safety team continuously validate them during the development life-cycle.

Also related to construct validity, we evaluated ImpRec without filtering any issue reports, i.e., we simply tried the RSSE by simulating the true historical inflow of issue reports. In a real setting, it is likely that ImpRec would only be used for defect reports that require corrective maintenance work. However, the decision to include all types of issue reports in the static validation, as long as they had an

attached CIA report, is unlikely to have improved our results. Instead, we suspect that the correctness of ImpRec might have been better if only defect reports were studied.

External validity threats are also substantial in relation to RQ1. The static validation procedure was designed to obtain preliminary results, and allow fast transition to the dynamic validation, the integral evaluation of this study. This choice lead to some simplifications of the static validation. Even though we simulate 1.5 years of true issue inflow, we study only one test set. Thus, it is possible that the results would have been different if we studied other test sets. An alternative design would have been to use k-tail evaluation [331], a type of k-fold cross validation preserving the internal order of elements. However, the extent of our static validation is in line with most previous work reported in Section 2.3. Moreover, we argue that a more thorough *in silico* evaluation is beyond the scope of this case study. We primarily consider correctness (RQ1) to be a prerequisite to study utility (RQ2), a quality characteristic we consider more interesting. Previous studies on the other hand, often use correctness as a proxy for utility, and leave studies with real users as important future work.

7.2 Dynamic Validation (RQ2 and RQ3)

Even though we carefully designed our study, e.g., by creating and evolving a detailed case study protocol as proposed by Runeson *et al.* [413], there are threats to construct validity. When conducting interviews, there is a risk that academic researchers and practitioners use different terminology and have different frames of reference. There is also a risk that the interview guides did not capture all utility aspects of ImpRec. To reduce these risks, the interview instruments were reviewed by all authors, and the interviewees were all given opportunity to discuss experiences openly. Furthermore, we improved construct validity by performing method triangulation, i.e., SLA and interviews.

The user log files did not always contain all information needed for proper SLA. Our design relied on that the participants used the confirmation functionality, and sometimes they did not. However, most ImpRec sessions were concluded with a click on the confirmation button (cf. D in Fig. 2), indicating that the participants followed our instructions. Finally, there is a risk that some participants refrained from using ImpRec because of the detailed user log files. To mitigate this threat, we presented the location of the log file, and explained that all data was stored without encryption in a readable XML format.

Our initial plan was to complement the interviews and the SLA by quantitative measurements in the issue tracker representing: 1) the quality of the CIAs, and 2) the effort required to conduct CIAs. Thus, we developed TimeCIA and ModCIA (presented in Section 5.3), and extracted the corresponding data prior to the initial interviews. However, the interviewees explained that there were too many confounding factors to interpret the two measures in meaningful ways. Consequently,

we did not attempt to triangulate the utility of ImpRec with these quantitative measures. Nonetheless, we argue that the interviews and the SLA suffice to answer RQ2 with reasonable validity.

Threats to internal validity deal with casual relations and confounding factors. Regarding RQ2, it is possible that the development phase in the organization affected the dynamic validation results. To study the utility of a CIA tool, its deployment should coincide with a CIA intensive phase. We deployed ImpRec in Unit Sweden after a formal verification phase, typically resulting in a subsequent peak of corrective maintenance, i.e., issue triaging and CIA. On the other hand, the point in time for deployment in Unit India was selected for convenience. The higher number of ImpRec uses in Unit Sweden reflects the different development phases during the data collection.

RQ3 addresses differences between project newcomers and seasoned developers. All seasoned developers are also senior, and there is a risk that seniors have a more balanced view on tools, i.e., seniors might be biased to more sceptical assessments of the utility of ImpRec. Another systematic bias in our study might come from cultural differences [420]. It is possible that the participants in the two units of analysis were culturally inclined to provide certain feedback. However, we consider the impact of this threat to be tolerable, as both Unit Sweden and Unit India reported positive and negative feedback. Similar bias may also apply to RQ2.

We discuss external validity both in terms of generalization to other developers in the organization, and to other companies active in safety-critical software development. We studied developers from two development teams in an organization comprising roughly 10 teams. Looking at the project history, the two selected teams have conducted CIAs with an average frequency, and the participants studied within the two teams represent different perspectives. Furthermore, we study developers working on two different continents. We find it likely that developers also from other teams in the organization, if they are working on evolving parts of the system covered by the knowledge base, could benefit ImpRec as well.

Regarding generalization to other companies, analytical generalization is required to relate our findings to other cases [413]. Our case study is an in-depth study of a specific organization, combining quantitative and qualitative analysis. Several aspects are unique to the case under study, e.g., the CIA template, and the practice of storing CIA reports as free text attachments in the issue tracker. It is important to mention that other organizations developing safety-critical systems may have other adaptations of their development processes to fulfil safety standards such as IEC61511-1 [238], ISO26262 [242], and EN50128 [174]. Still, explicit CIAs are required in all organizations modifying a safety certified software system [68]. Moreover, our cross-domain survey of CIA also suggests that while details differ, many of the CIA challenges are universal in safety-critical development [136]. As such, providing in-depth industrial case studies can enable important knowledge transfer between domains, a phenomenon also reported by participant B as remarkably limited at the moment.

The analysis of qualitative research lies in interpretation by the involved researchers, thus exact replications are improbable, which constitutes a threat to reliability. However, to increase the reliability of our study, we combined the interviews with SLA. Another threat to *in situ* studies, particularly with longitudinal data collection, is that strong relationships are created between researchers and study participants. Other researchers might have developed different relations, influencing the interviews in other ways.

Finally, there are multiple quality attributes available for RSSE evaluations. Avazpour *et al.* listed 16 attributes [32], but this study focused only on correctness and utility. Other researchers would maybe select other attributes. From our perspective, the most important ones among the 14 attributes we did not evaluate are coverage and risk. However, we discuss them both in Section 8.

8 Discussion

In this section, we first discuss the research questions presented in Section 5. Second, we discuss our work in the light of previous research on CIA. Third, we discuss the implications for industry practice.

8.1 Revisiting the Research Questions

RQ1 addresses the correctness of ImpRec [32], a question that we tackled using *in silico* simulation and quantitative IR measures, in what we refer to as static validation [201]. We studied two configurations of ImpRec, and conclude that *ImpRec recommends about 30% of the true impact among the top-5 items and 40% among the top-10 items.* Furthermore, while ImpRec A recommends few truly impacted artifacts after rank 10, *ImpRec B exceeds 50% among the top-20 items.*

The ranking function of ImpRec performed better than two naïve approaches to automated CIA: 1) recommending artifacts impacted by textually similar issue reports, and 2) recommending the most frequently impacted artifacts. Furthermore, the *in situ* evaluation showed that the participants considered the ranking function to be helpful, as indicated by the click distribution and as reported in the post-study interviews.

To the best of our knowledge, ImpRec is the first tool for automated CIA that exclusively targets non-code software artifacts. While this means that comparisons to previous work should only be made to observe general trends, we conclude that the correctness of ImpRec is in line with what can be found in the scientific literature. ImpRec obtains better recall than precision, but we agree with previous researchers [118, 144, 231] and consider recall more important (as long as the total number of recommendations is still reasonable), we postulate that *the correctness of ImpRec is good enough to commence dynamic validation.*

RQ2 deals with the utility of ImpRec [32], and especially whether developers recognized the value of the RSSE. We assessed this using dynamic validation [201], by deploying ImpRec in two development teams, and then collecting data in a longitudinal *in situ* study. While we also collected quantitative measures, we primarily discuss utility based on the post-study interviews using the QUPER model [390].

The post-study interviews with Unit Sweden suggest that the correctness of the RSSE has passed the utility breakpoint, i.e., *ImpRec is useful in supporting CIA*. Participants D and G estimated the utility breakpoint to be at 25% and 10% recall, respectively, strictly lower than ImpRec's recall at 40% already at N=10. Interestingly, participant G put the differentiation breakpoint at 25% and the saturation breakpoint at 50%, as he does not want a tool to deliver "too much". Several other participants, including Unit India, also reported positive feedback from working with ImpRec.

The participants reported two considerations related to utility that will impact future work. First, participant D expressed concerns that ImpRec rarely reported anything that he did not already know. This implies that there is a risk that *ImpRec might primarily deliver obvious recommendations*, i.e., only reinforcing what the developers already know, but providing limited novelty [402]. On the other hand, delivering confirmation is critical for an RSSE to establish user trust [32]. Still, future work needs to further assess the value of the true recommendations provided by ImpRec. In the same vein, participant E warned that *ImpRec risks propagating errors from previous CIA reports*. This is indeed true, and further research is required on how to adapt development processes to take such error propagation into account (see also Section 8.3).

Participants E and F emphasized another aspect of utility: they claimed that *an RSSE like ImpRec must be integrated in the existing tool chain*, and not executed as a separate tool. We were well aware of this during design of the tool [69], but as the organization was (and still is) in a transition to replace the current issue tracker, we decided to integrate ImpRec in another existing support tool instead only (IA Sidekick). However, as most developers were unaware of IA Sidekick, it did not support the dissemination of ImpRec much.

RQ3 explores whether project newcomers value navigational tool support more than developers that are more knowledgeable, as suggested by previous work on tool support for software engineering [127, 366]. In our *in situ* study, we consider Participants (F), (G), (J), and (K) as newcomers with less knowledge of both the domain in general and the particular software system.

Participant F obtained an RcImp matching the static validation (cf. Table 4, but the other newcomers obtained modest results in terms of recall. On the other hand, *the newcomers confirmed a slightly higher number of useful related issue reports* than the seasoned developers (on average 1.7 per search session, compared to 1.3) but the difference is not statistically significant. The average number of confirmed impacted artifacts however was practically the same for newcomers and seasoned

developers.

Newcomers expressed some of the most positive comments. Participants J and K in Unit India were both very positive to ImpRec and interested in the future evolution of the tool. Moreover, participant G in Unit Sweden put the utility breakpoint in the QUPER model well below the current correctness of ImpRec. The newcomers appeared to particularly value the quick access to previous issue report provided by the RSSE, i.e., the qualitative feedback suggest that *introducing state-of-the-art search functionality in the issue tracker is promising*.

The seasoned developers expressed most of the risks involved in increasing the level of automation in CIA. While the junior developers (G), (J), and (K) did not discuss any risks involved in tool support during interviews, some senior developers were more defensive. Participant D warned that mistakes might be propagated using tools. Participant H did not see obvious use cases for ImpRec, and participant E questioned the lack of novelty in the ImpRec recommendations. To conclude, the qualitative feedback suggests that *newcomers are more positive to navigational support provided by an RSSE compared to seasoned developers*. However, while our results slightly indicate that newcomers particularly appreciate recommendations of related issue reports, *whether newcomers in general benefit more from ImpRec than seasoned developers remains inconclusive*.

8.2 ImpRec and State-of-the-Art Research

Our study brings several novel contributions to CIA and RSSE research, in particular from an empirical perspective. In this section, we discuss four aspects where our work goes further than previous studies. We mainly compare our work to previous CIA research based on the taxonomy of CIA support proposed by Lehnert [293] and his accompanying literature review [292]. Table 5 summarizes our work according to the taxonomy. We conclude the section by sharing some lessons learned that might help advance studies on CIA.

First, although CIA is fundamental in safety standards (e.g., [174, 242, 382]), most of the previous evaluations of tool support for CIA exclusively consider Open Source Software (OSS) [292]. We have previously identified the same lack of studies in proprietary contexts regarding tool support for traceability management [76], another corner stone in safety-critical development [115, 210].

Exclusively focusing on OSS is unfortunate, since there is a need for empirical studies on safety-critical software development in industry [350]. Also, there are still no large safety-critical OSS systems available as surrogates [441], thus researchers must target proprietary systems. The over-representation of studies in the OSS domain applies to software engineering research in general, explained by the appealing availability of large amounts of data and the possibility of replications [404]. Thus, this study is a rare example of an in-depth empirical study on tool support for CIA, tailored for a specific safety-critical development process, in a proprietary context.

Table 5: ImpRec compared to Lehnert's taxonomy of CIA tools [292]. Comments in italic font represent types that do not exist in the original taxonomy.

Criterion	**Comment**
Scope of Analysis	Misc. artifacts
Utilized Technique(s)	History mining, Traceability, Information retrieval
Granularity of - Entities - Changes - Results	 *Document* *Issue report* *Document*
Style of Analysis	Search based
Tool Support	ImpRec
Supported Languages	N/A
Scalability	*Full scale*
Experimental Results - Size - Precision - Recall - Time	 $\approx$ 50,000 entities Pr@10=0.05 Rc@10=0.40 < 1 s per search + *in situ evaluation*

Second, we conducted a longitudinal *in situ* study. Most previous evaluations on CIA tools were instead only evaluated *in silico*, i.e., they were assessed based on tool output from experimental runs on a computer. No study included in Lehnert's literature review involved a prolonged analysis of a deployed tool [292]. In the RSSE community, Robillard and Walker recently highlighted the lack of studies including humans in the loop [402]. Our study is unique in its *in situ* design, i.e., we study how developers in Sweden and India interact with ImpRec in their own work environment, while they work on real CIA. Future research should continue with this more holistic approach, by deploying tools and studying human output as well as tool output.

Third, ImpRec addresses CIA of non-code artifacts. A clear majority of previous studies on CIA target the source code level, but our work is instead among the few exceptions that deals with miscellaneous artifacts. In safety-critical software development, the impact on different artifacts must be analyzed. Our previous cross-domain survey of practitioners working with CIA in safety-critical systems shows that not only source code impact is difficult to analyze, but also other development artifacts [136]. The initial interviews in this case study confirm the importance of extending CIA support to misc. types of artifact. In fact, some of the developers even stated that tool support for CIA among requirements, test cases and related documentation is *more* important than tool support for source

code CIA, as they find it more difficult (and less interesting) to stay on top of information that does not reside in the source code repository, see Section 3.2. We suspect that one reason for the strong code-orientation in previous CIA research originates from the OSS domain in which fewer types of software artifacts are typically maintained. Consequently, we argue that more CIA researchers should study software systems certified by some of the established safety standards in industry, to enable more studies beyond the source code level.

Fourth, ImpRec combines techniques proposed in previous work in a novel way. Lehnert's taxonomy contains 10 different techniques to support CIA [293]. As presented in Table 5, ImpRec combines a collaboratively created knowledge base with a state-of-the-art search solution. The knowledge base is established using History Mining (HM) of previous trace links, i.e., Traceability (TR). Apache Lucene provides the Information Retrieval (IR) part, the driver of the ImpRec approach. According to Lehnert's literature review, HM+TR have been combined in previous work [135], as well as HM+IR [91, 92, 259] and TR+IR [464]. However, ImpRec is the only tool support for CIA that combines HM+TR+IR.

Regarding the remaining criteria in Lehnert's taxonomy [293], more aspects of ImpRec are worth mentioning. The granularity considered in our work is 'documents', with 'issue reports' as change triggers. Our RSSE is used for a search based style of CIA, i.e., on demand per issue report, the least frequent style in Lehnert's literature review [292]. The criterion 'supported languages' does not apply to ImpRec, as it does not deal with source code impact. Finally, the scalability of our approach is implicit, as we already used it *in situ* in a real industrial setting. The searches are still quick, and we leverage on size, i.e., we expect a larger knowledge base to bring higher correctness and utility.

Finally, we report some lessons learned that might support future studies on CIA. CIA analysis is a complex cognitive task that requires much from the developer. The initial interviews in this study clearly shows that assessing the value of tool support cannot be made using simple measures such as TimeCIA and ModCIA (introduced in Section 5.3), as there are too many confounding factors. Furthermore, our post-study interviews reveal that also correctness (in terms of IR measures) is too simplistic. The value of recommendations is not binary; a high amount of correct results might still be barely useful, while a single correct item (if delivered with a high ranking) can bring the experienced utility to high levels. In conclusion, our work stresses the importance of qualitative analysis of output from CIA tools.

8.3 ImpRec and Implications for Industry Practice

In this section we discuss the findings most important to industry practice. While there are several aspects that could be of interest to industry practitioners in safety-critical development contexts, we focus on three topics: 1) wasted effort when

working with an inadequate issue tracker, 2) the potential of CIA history mining, and 3) challenges in introducing CIA tool support.

First, we found *strong evidence that using an underdeveloped issue tracker impedes both CIA and issue management in general*. The majority of participants in the study were critical about the currently used issue tracker, explaining that it was slow, and poor at both searching and browsing issue reports. Based on the feedback functionality of ImpRec, we also identified that *many relations among issue reports were not properly stored in the issue tracker*. It might be worthwhile to oversee how and when relations are specified, as research has shown that helping other developers quickly find all related issues speeds up maintenance work [53]. In our study, *especially the junior developers appreciated quick access to previous issue reports*. Our recommendation to industry is to at least *introduce proper search functionality in the issue tracker* (e.g., using the OSS library Apache Lucene as we did in ImpRec) as it would constitute a small effort with a promising return on investment. Furthermore, future search solutions might turn more accurate if the inter-issue relations are properly stored, as network measures are central in state-of-the-art ranking functions.

Second, our work highlights the significant potential of what mining explicit knowledge from historical CIA reports could bring. As presented in Section 3.2, one of the challenges of the rigid CIA process in safety-critical development is that developers view it as an activity that simply must be done to achieve safety certification, but that the CIA reports once completed bring them no personal value. Several participants in our study confirm this view, thus industry should make an effort to increase the appeal of CIA. We argue that *letting developers reuse knowledge captured in previous CIAs could make developers more inclined to write high quality CIA reports, and to make them living documents*. Developers take pride in evolving source code to high standards, but the CIA report is primarily a one-shot production, and developers rarely look back. Spending hundreds of hours on CIA reports in a project merely to please external certification agencies could be considered as waste unless the knowledge built during the process is reused to improve the software development.

Even though our study shows that reusing previous CIA reports has the potential to support developers, we are aware that our approach has limitations. Knowledge reuse is only possible if the project contains enough history, and from a collaboratively created knowledge base, an RSSE can only recommend already encountered artifacts. This aspect is referred to as the *coverage* of the RSSE [32]. Table 4 shows that ImpRec's coverage varies across different parts of the system, and for participants working on new parts of the system ImpRec brings little value. However, based on our positive assessment of utility (RQ2), we infer that *mining a knowledge base from historical CIA reports reaches useful coverage by focusing on the most volatile components*.

Third, introducing new tool support in a large organization is a challenging endeavour. Dubey and Hudepohl share some experiences in a recent publica-

tion [164], in which they categorize the challenges along three dimensions: 1) technical, 2) collaboration, and 3) motivational.

Technical challenges originate in the varied environments encountered in large organizations. ImpRec is tailored for a specific issue tracker, and a rigorous CIA process using a formal CIA template. Thus, the use of ImpRec is limited to certain parts of the organization. *Collaboration challenges* on the other hand deal with communication issues between the tool supplier and the users. As suggested by Dubey and Hudepohl [164], we created an easy-to-follow user manual and provided support via email. The *motivational dimension* is probably the most protruding among Dupey and Hudepohl's categories. Working with ImpRec must be better than manual CIAs, otherwise the RSSE will have no users. Participants in our study particularly stressed the importance of seamless integration in the issue tracker and fast searches.

There seems to be an additional dimension not mentioned by Dubey and Hudepohl [164], which applies to introducing tool support in an environment certified for safety-critical development: the *organizational dimension*. To formally introduce a new tool in the organization under study, a 'tool selection report' must be authored. The report should explain to the external safety assessor how system safety might be affected with the new tool. Since ImpRec targets CIA, a safety-critical activity, also development processes must be adapted. While ImpRec has the ability to identify impacted artifacts that could be missed otherwise, the tool might also lull the developers to a false sense of security. Thus, the CIA process guidelines would have to be updated to counter this phenomenon. Future research should further explore how RSSEs could be introduced in safety-critical contexts.

9 Conclusion and Future Work

This paper reports from an industrial case study on a Recommendation System for Software Engineering (RSSE) for Change Impact Analysis (CIA) called ImpRec. We introduced ImpRec [75] to provide decision support by presenting potentially impacted non-code artifacts, tailored for a particular development organization. ImpRec recommendations originate in the textual content of an incoming issue report, and uses network analysis in a collectively created knowledge base to compute candidate impact.

We evaluate ImpRec in a two-step study, as recommended practice for technology transfer by Gorschek *et al.* [201]. First, we conducted static validation *in silico*, to assess the correctness of ImpRec (RQ1). Our results suggest that ImpRec presents about 40% of the true impact within the top-10 recommendations. Second, we conducted dynamic validation *in situ*, by deploying ImpRec in two development teams for several months. We assess the utility of ImpRec (RQ2) based on collected user logs and interviews. Our results indicate that ImpRec's current level of correctness represents tool support that has passed the utility breakpoint, i.e.,

the developers recognize the value of using the tool. Also, developers acknowledge the overall approach of reusing knowledge from past CIA to provide decision support, and they are positive to further research.

Our findings have implications for both research and practice. First, our study contributes to the state-of-the-art of RSSE evaluation. Our case study is a rare example of an in-depth evaluation of an RSSE in a proprietary context. ImpRec provides novel CIA tool support by solely targeting non-code impact, a type of impact stressed as particularly challenging by our interviewees. Second, regarding implications for industry, we show that if an issue tracker does not offer adequate search and navigation, it impedes both CIA and issue management in general. We argue that simply storing highly accurate CIA reports in a database, without motivating developers to benefit from the captured knowledge, is a waste of effort. By conducting link mining, the knowledge in the historical CIA reports can be used to provide decision support that might be especially helpful for junior developers.

While the current version of ImpRec appears to be mature enough to be used in industry, there are several important paths for future work. First, the RSSE itself should be further evolved. The correctness might be increased by attempting to improve preprocessing and to tune internal parameters. A promising direction would also be to introduce source code impact to ImpRec. While it is not considered the highest priority by the practitioners, it might result in a more complete semantic network, thus offering more accurate recommendations. Second, as ImpRec uses the historical CIA reports to build its knowledge base, we expect improvements as more data becomes available. However, the knowledge base might also turn partly obsolete, thus decreasing the correctness of ImpRec. Future work should investigate how to maintain a deployed RSSE in industry, with regard to retraining when additional data becomes available and monitoring performance as training data becomes older. ImpRec should also be improved along those lines, as the current version requires manual creation of the knowledge base, instead of online learning [57]. Finally, research must be directed at how to introduce additional tool support in safety-critical contexts, in line with work by Dupey and Hudepohl [164]. Deployment of new tools always introduces risks, and the mitigation strategies in the target organizations should involve both adapted processes and practices.

Acknowledgements

This work was funded by the Industrial Excellence Center EASE - Embedded Applications Software Engineering[5]. Thanks go to all participants in the case study.

[5] http://ease.cs.lth.se

PART III: THE UTILIZATION PHASE

CHAPTER V

TUNER: A FRAMEWORK FOR TUNING SOFTWARE ENGINEERING TOOLS WITH HANDS-ON INSTRUCTIONS IN R

Abstract

Numerous tools automating various aspects of software engineering have been developed, and many of the tools are highly configurable through parameters. Understanding the parameters of advanced tools often requires deep understanding of complex algorithms. Unfortunately, sub-optimal parameter settings limit the performance of tools and hinder industrial adaptation, but still few studies address the challenge of tuning software engineering tools. We present TuneR, an experiment framework that supports finding feasible parameter settings using empirical methods. The framework is accompanied by practical guidelines of how to use R to analyze the experimental outcome. As a proof-of-concept, we apply TuneR to tune ImpRec, a recommendation system for change impact analysis. Compared to the output from the default setting, we report a 20.9% improvement in the response variable. Moreover, TuneR reveals insights into the interaction among parameters, as well as non-linear effects. TuneR is easy to use, thus the framework has potential to support tuning of software engineering tools in both academia and industry.

Markus Borg *Submitted to a journal*

1 Introduction

Tools that increase the level of automation in software engineering are often highly configurable through parameters. Examples of state-of-the-art tools that can be configured for a particular operational setting include EvoSuite for automatic test suite generation [187], FindBugs for static code analysis [34], and MyLyn, a task-oriented recommendation system in the Eclipse IDE [269]. However, the performance of these tools, as well as other tools providing decision support, generally depends strongly on the parameter setting used [45], often more so than the choice of the underlying algorithm [291]. The best parameter setting depends on the specific development context, and even within the same context it might change over time.

Finding feasible parameter settings is not an easy task. Automated tools in software engineering often implement advanced techniques such as genetic algorithms, dimensionality reduction, Information Retrieval (IR), and Machine Learning (ML). Numerous studies have explored how tool performance can be improved by tailoring algorithms and tuning parameters, for example in test data generation [188], test case selection [305], fault localization [4, 446], requirements classification [98], and trace recovery [360, 500]. We have previously published a systematic mapping study highlighting the data dependency of IR-based trace recovery tools [74], and Hall *et al.* found the same phenomenon in a systematic literature review on bug prediction, stating that "*models perform the best where the right technique has been selected for the right data, and these techniques have been tuned for the model rather than relying on default tool parameters*" [212]. However, the research community cannot expect industry practitioners to have the deep knowledge required to fully understand the settings of advanced tools.

Feasible tuning of parameter settings is critical for successful transfer of Software Engineering (SE) tools from academia to industry. Unfortunately, apart from some work on Search-Based Software Engineering (SBSE) [25, 178] there are few software engineering publications that specifically address parameter tuning. One could argue that academia should develop state-of-the-art tools, and that the actual deployment in different organizations is simply a matter of engineering. However, we argue that practical guidelines for tuning SE tools, i.e., finding feasible parameter settings, are needed to support adaptation to industrial practice.

In this paper we discuss ImpRec [75], a Recommendation System for Software Engineering (RSSE) [402] developed to support Change Impact Analysis (CIA) in a company developing safety-critical software systems. ImpRec implements ideas from the area of Mining Software Repositories (MSR) to establish a semantic network of dependencies, and uses state-of-the-art IR to identify textually similar nodes in the network. The tool combines the semantic network and the IR system to recommend artifacts that are potentially impacted by an incoming issue report, and presents a ranked list to the developer. During development of the tool, we had to make several detailed design decisions, e.g., "how should distant artifacts

in the system under study be penalized in the ranking function?" and "how should we weigh different artifact features in the ranking function to best reflect the confidence of the recommendations?". Answering such questions at design time is not easy. Instead we parametrized several decisions, a common solution that effectively postpones decisions to the tool user. We have deployed an early version of ImpRec in two pilot development teams to get feedback [80]. However, we did not want to force the study participants to consider different parameter settings; instead we deployed ImpRec with a default setting based on our experiences. The question remains however; is the default setting close to the optimum?

We see a need for tuning guidelines for SE tools, to help practitioners and applied researchers to go beyond trial and pick-a-winner approaches. We suspect that three sub-optimal tuning strategies [165, pp. 211] [342, pp. 4] dominate tuning of SE tools: 1) *ad hoc* tuning, 2) quasi-exhaustive search, and 3) Change One Parameter at a Time (COST) analysis. *Ad hoc tuning* might be a quick way to reach a setting, but non-systematic tuning increases the risk of deploying tools that do not reach their potential, therefore not being disseminated properly in industry. *Quasi-exhaustive search* might be possible if the evaluation does not require too much execution time, but it does not provide much insight in the parameters at play unless the output is properly analyzed. *COST analysis* is a systematic approach to tuning, but does not consider the effect of interaction between parameters.

We present TuneR, a framework for tuning parameters in automated software engineering tools. The framework consists of three phases: 1) Prepare Experiments, 2) Conduct Screening, and 3) Response Surface Methodology. The essence of the framework lies in space-filling and factorial design, established methods to structure experiments in Design of Computer Experiments (DoCE) and Design of Experiments (DoE), respectively. As a proof-of-concept, we apply TuneR to find a feasible parameter setting for ImpRec. For each step in TuneR, we present hands-on instructions of how to conduct the corresponding analysis using various packages for R [381], and the raw data is available on the companion website[1]. Using TuneR we increase the accuracy of ImpRec's recommendations, with regard to the selected response variable, by 20.9%. We also validate the result by comparing the increased response to the outcome of a more exhaustive space-filling design.

The rest of this paper is structured as follows: Section 2 introduces the fundamental concepts in DoE and DoCE, and discusses how tuning of SE tools is different. Section 3 presents related work on finding feasible parameter setting for SE tools. In Section 4 we introduce ImpRec, the target of our tuning experiments. The backbone of the paper, the extensive presentation of TuneR, interweaved with the proof-of-concept tuning of ImpRec, is found in Section 5. In Section 6, we report from the exhaustive experiment on ImpRec parameter settings. Section 7 discusses our results, and presents the main threats to validity. Finally, Section 8 concludes the paper.

[1] http://serg.cs.lth.se/research/experiment-packages/tuner/

2 Background

This section introduces design of experiments, both of physical and simulated nature, and presents the terminology involved. Then we discuss how tuning of automated SE tools differ from traditional experiments. We conclude the section by reporting related work on experimental frameworks and parameter tuning in software engineering.

2.1 Design of Experiments

Design of Experiments (DoE) is a branch of applied statistics that deals with planning and analyzing controlled tests to evaluate the factors that affect the output of a process [342]. DoE is a mature research field, a key component in the scientific method, and it has proven useful for numerous engineering applications [236]. Also, DoE is powerful in commercialization, e.g., turning research prototypes into mature products ready for market release [351]. DoE is used to answer questions such as "what are the key factors at play in a process?", "how do the factors interact?", and "what setting gives the best output?".

We continue by defining the fundamental experimental terminology that is used throughout the paper. For a complete presentation of the area we refer to one of the available textbooks, e.g., Montgomery [342], Box *et al.* [82], and Dunn [165]. An *experiment* is a series of *experimental runs* in which changes are made to input variables of a system so that the experimenter can observe the output *response*. The input variables are called *factors*, and they can be either *design factors* or *nuisance factors*. Each design factor can be set to a specific *level* within a certain *range*. The nuisance factors are of practical significance for the response, but they are not interesting in the context of the experiment.

Dealing with nuisance factors is at the heart of traditional DoE. Nuisance factors are classified as *controllable*, *uncontrollable*, or *noise factors*. Controllable nuisance factors can be set by the experimenter, whereas uncontrollable nuisance factors can be measured but not set. Noise factors on the other hand can neither be controlled nor measured, and thus require more of the experimenter.

The cornerstones in the experimental design are *randomization*, *replication*, and *blocking*. Randomized order of the experimental runs is a prerequisite for statistical analysis of the response. Not randomizing the order would introduce a systematic bias into the responses. Replication means to conduct a repeated experimental run, independent from the first, thus allowing the experimenter to estimate the experimental error. Finally, blocking is used to reduce or eliminate the variability introduced by the nuisance factors. Typically, a block is a set of experimental runs conducted under relatively similar conditions.

Montgomery lists five possible goals of applying DoE to a process: 1) factor screening, 2) optimization, 3) confirmation, 4) discovery, and 5) robustness [342, pp. 14]. *Factor screening* is generally conducted to explore or characterize a new

process, often aiming at identifying the most important factors. *Optimization* is the activity of finding levels for the design factors that produce the best response. *Confirmation* involves corroborating that a process behaves in line with existing theory. *Discovery* is a type of experiments related to factor screening, but the aim is to systematically explore how changes to the process affect the response. Finally, an experiment with a *robustness* goal tries to identify under which conditions the response substantially deteriorates. As the goal of the experiments conducted in this paper is to find the best response for an automated software engineering tool by tuning parameters, i.e., optimization, we focus the rest of this section accordingly.

The traditional DoE approach to optimize a process involves three main steps: 1) factor screening to narrow down the number of factors, 2) using *factorial design* to study the response of all combinations of factors, and 3) applying *Response Surface Methodology* (RSM) to iteratively change the setting toward an optimal response [347]. Factorial design enables the experimenter to model the response as a *first-order model* (considering *main effects* and *interaction effects*), while RSM also introduces a *second-order* model in the final stage (considering also quadratic effects).

Different experimental designs have been developed to study how design factors affect the response. The fundamental design in DoE is a factorial experiment, an approach in which design factors are varied together (instead of one at a time). The basic factorial design evaluates each design factor at two levels each, referred to as a 2^k factorial design. Such a design with two design factors is represented by a square, where the corners represent the levels to explore in experimental runs (see A in Fig. 1). When the number of design factors is large, the number of experimental runs required for a full factorial experiment might not be feasible. In a *fractional factorial experiment* only a subset of the experimental runs are conducted. Fractional factorial designs are common in practice, as all combinations of factors rarely need to be studied. The literature on fractional factorial designs is extensive, and we refer the interested reader to discussions by Montgomery [342] and Dunn [165].

All points in the experimental designs represent various levels of a design factor. In DoE, all analysis and model fitting are conducted in *coded units* instead of in *original units*. The advantage is that the model coefficients in coded units are directly comparable, i.e., they are dimensionless and represent the effect of changing a design factor over a one-unit interval [342, pp. 290]. We use 1 and -1 to represent the high and low level of a design factor in coded units.

Factorial design is a powerful approach to fit a first-order model to the response. However, as the response is not necessarily linear, additional experimental runs might be needed. The first step is typically to add a *center point* to the factorial design (cf. B in Fig. 1). If quadratic effects are expected, e.g., indicated by experimental runs at the center point, the curvature needs to be better characterized. The most popular design for fitting a second-order model to the response is the *Central Composite Design* (CCD) [342, pp. 501] (cf. C in Fig. 1). CCD

complements the corners of the factorial design and the center point with axial points. A CCD is called *rotatable* if all points are at the same distance from the center point [276, pp. 50].

RSM is a sequential experimental procedure for optimizing a response (for a complete introduction we refer the reader to Myers' textbook [347]). In the initial optimization phase, RSM assumes that we operate at a point far from the optimum condition. To quickly move toward a more promising region of operation, the experimenter fits a first-order model to the response. Then, the operating conditions should be iteratively changed along the *path of steepest ascent*. When the process reaches the region of the optimum, a second-order model is fitted to enable an analysis pinpointing the best point.

DoE has been a recommended practice in software engineering for decades. The approaches have been introduced in well-cited software engineering textbooks and guidelines, e.g., Basili *et al.* [43], Pfleeger [376], and Wohlin *et al.* [477]. However, tuning an automated software engineering tool differs from traditional experiments in several aspects, as discussed in the rest of this section.

2.2 Design of Computer Experiments

DoE was developed for experiments in the physical world, but nowadays a significant amount of experiments are instead conducted as computer simulation models of physical systems, e.g., during product development [288]. Exploration using computer simulations shares many characteristics of physical experiments, e.g., each experimental run requires input levels for the design factors and results in one or more responses that characterize the process under study. However, there are also important differences between physical experiments and experiments in which the underlying reality is a mathematical model explored using a computer.

Randomization, replication, and blocking, three fundamental components of DoE, were all introduced to mitigate the random nature of physical experiments. Computer models on the other hand, unless programmed otherwise, generate deterministic responses with no random error [463]. While the deterministic responses often originate from highly complex mathematical models, repeated experimental runs using the same input data generates the same response, i.e., replication is not required. Neither does the order of the experimental runs need to be randomized, nor is blocking needed to deal with nuisance factors. Still, assessing the relationship between the design factors and the response in a computer experiment is not trivial, and both the design and analysis of the experiment need careful thought.

Design of Computer Experiments (DoCE) focuses on *space-filling designs*. Evaluating only two levels of a design factor, as in a 2^k factorial design, might not be appropriate when working with computer models, as it can typically not be assumed that the response is linear [176, pp. 11]. Instead, interesting phenomena can potentially be found in all regions of the experimental space [342, pp. 524].

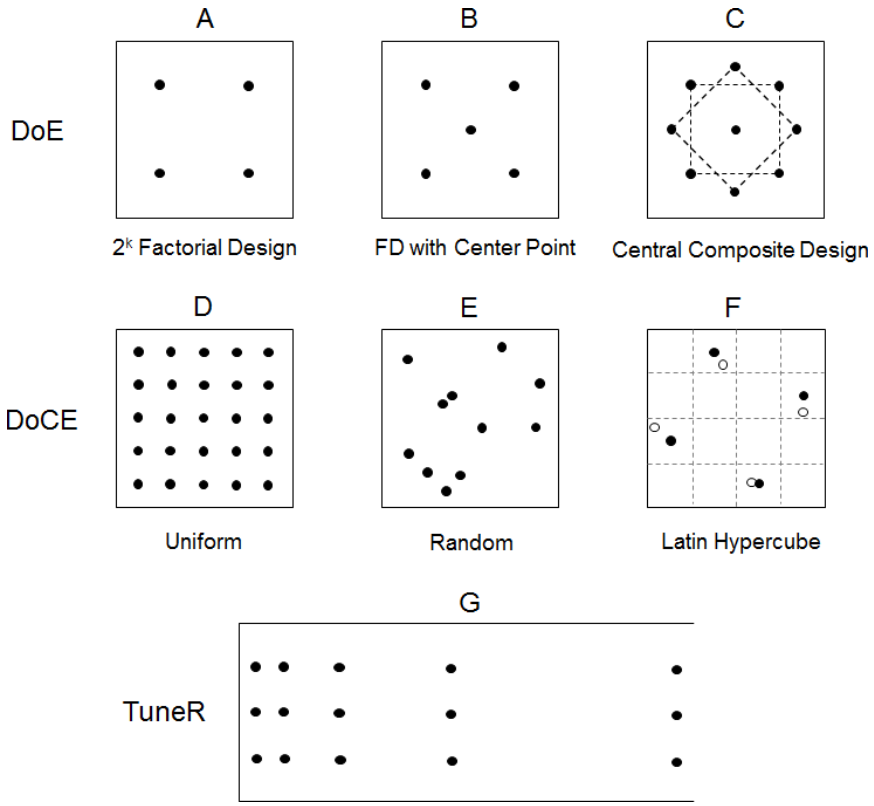


Figure 1: Overview of experimental designs for two factors. Every point represents an experimental setting.

The simplest space-filling designs are *uniform design* (cf. D in Fig. 1), in which all design points are spread evenly, and *random design* (cf. E in Fig. 1). Another basic space-filling design is the *Latin Hypercube design*. A two-factor experiment has its experimental points in a latin square if there is only one point in each row and each column (cf. F in Fig. 1), in line with the solution to a sudoku puzzle. A Latin Hypercube is the generalization to an arbitrary number of dimensions. Latin Hypercubes can be combined with randomization to select the specific setting in each cell, as represented by white points in Figure 1.

Also RSM needs adaptation for successful application to computer experiments. There are caveats that need to be taken into consideration when transferring RSM from DoE to DoCE. Vining highlights that the experimenter need some information about starting points, otherwise there is a considerable risk that RSM ends up in a local optimum [463]. Moreover, bumpy response surfaces, which computer

models might generate, pose difficulties for optimization. Consequently, a starting point for RSM should be in the neighborhood of an acceptable optimum. Finally, RSM assumes that there should be only few active design factors. Vining argues that both starting points and the number of design factors should be evaluated using screening experiments [463], thus screening is emphasized as a separate phase in TuneR.

2.3 Tuning Automated Software Engineering Tools

DoE evolved to support experiments in the physical world, and DoCE was developed to support experiments on computer models of physical phenomena. The question whether software is tangible or intangible is debated from both philosophical and juridical perspectives (see e.g., [50, 343]), but no matter what, there are differences between software and the entities that are typically explored using DoE and DoCE. Furthermore, in this paper we are interested in using experiments for tuning[2] a special type of software: tools for automated software engineering. We argue that there are two main underlying differences between experiments conducted to tune automated SE tools and traditional DoCE. First, automated SE tools are not computer models of an entity in the physical world. Thus, we often cannot relate the meaning of various parameter settings to characteristics that are easily comprehensible. In DoCE however, we are more likely to have a preunderstanding of the characteristics of the underlying physical phenomenon. Second, a tuned automated SE tool is not the primary deliverable, but a means to an end. An automated SE tool is intended to either improve the software under development, or to support the ongoing development process [182]. In DoCE on the other hand, the simulation experiments tend to be conducted on a computer model of the product under development or the phenomenon under study.

Consequently, an experimenter attempting to tune an automated SE tool must consider some aspects that might be less applicable to traditional DoCE. The experimenter should be prepared for unexpected responses in all regions of the experiment space, due to the lack of connection between parameters and physical processes. Parameter ranges resulting in feasible responses might exist anywhere in the experiment space, thus some variant of space-filling designs need to be applied as in DoCE. However, responses from automated SE tools cannot be expected to behave linearly, as the response might display sudden steps in the response or asymptotic behavior. While certain peculiarities might arise also when calibrating physical processes, we believe that they could be more common when tuning automated SE tools. Other aspects that must be taken into consideration are execution time and memory consumption. An SE tool is not useful if it cannot deliver its output in a reasonable amount of time, and it should be able to do so with the memory available in the computers of the target environment.

[2] Adjusting parameters of a system is known as *calibration* when they are part of a physical process, otherwise the activity is called *tuning* [299].

When tuning an automated SE tool, we propose that it should be considered a black-box model (also recommended by Arcuri and Fraser [25]). We define a black-box model, inspired by Kleijnen, as "*a model that transforms observable input into observable outputs, whereas the values of internal variables and specific functions of the tool implementation are unobservable*" [276, pp. 16]. For any reasonably complex SE tool, we suspect that fully analyzing how all implementation details affect the response is likely to be impractical. However, when optimizing a black-box model we need to rely on heuristic approaches, as we cannot be certain whether an identified optimum is local or global. An alternative to heuristic approaches is to use metaheuristics (e.g., genetic algorithms, simulated annealing, or tabu search [15]), but such approaches require extensive tuning themselves.

The main contribution of this paper is *TuneR*, a heuristic experiment framework for tuning automated SE tools using R. TuneR uses a space-filling design to screen factors of a black-box SE, uniform for bounded parameters and a geometric sequence for unbounded parameters as shown in Figure 1 (G). Once a promising region for the parameter setting has been identified, TuneR attempts to apply RSM to find a feasible setting. We complement the presentation of TuneR with a hands-on example of how we used it to tune the RSSE ImpRec.

3 Related Work on Parameter Tuning in Software Engineering

Several researchers have published papers on parameter tuning in software engineering. As the internals of many tools for automated SE involve advanced techniques, such as computational intelligence and machine learning, academic researchers must provide practical guidelines to support knowledge transfer to industry. In this section we present some of the most related work on tuning automated SE tools. All tools we discuss implement metaheuristics to some extent, a challenging topic covered by Birattari in a recent book [61]. He reports that *most tuning of metaheurstics is done by hand and by rules of thumb*, showing that such tuning is not only an issue in SE.

Parameter tuning is fundamental in Search-Based Software Engineering (SBSE) [25, 188]. As SBSE is based on metaheuristics, its performance is heavily dependent on context-specific parameter settings. However, some parameters can be set based on previous knowledge about the problem and the software under test. Fraser and Arcuri refer to this as seeding, i.e., "*any technique that exploits previous related knowledge to help solve the testing problem at hand*" [188]. They conclude that *seeding is valuable in tuning SBSE tools*, and present empirical evidence that the more domain specific information that can be included in the seeding, the better the performance will be. In line with the recommendations by Fraser and Arcuri, we emphasize the importance of pre-understanding by including it as a separate step in TuneR.

Arcuri and Fraser recently presented an empirical analysis on how their tool EVOSUITE, a tool for test data generation, performed using different parameter settings [25]. Based on more than one million experiments, they show that *different settings cause very large variance* in the performance of EVOSUITE, but also that *"default" settings presented in the literature result in reasonable performance*. Furthermore, they find that *tuning EVOSUITE using one dataset and then applying it on others brings little value*, in line with the No Free Lunch Theorem by Wolpert and Macready [479]. Finally, they applied RSM to tune the parameters of EVOSUITE, but conclude that *RSM did not lead to improvements compared to the default parameter setting*. Arcuri and Fraser discuss the unsuccessful outcome of their attempt at RSM, argue that it should be treated as inconclusive rather than a negative result, and call for more studies on tuning in SE. Their paper is concluded by general guidelines on how to tune parameters. However, the recommendations are on a high-level, limited to a warning on over-fitting, and advice to partition data into non-overlapping training and test sets. The authors also recommend using 10-fold cross-validation in case only little data is available for tuning purposes. Our work on TuneR complements the recommendations from Arcuri and Fraser, by providing more detailed advice on parameter tuning. Also, there is no conflict between the two sets of recommendations, and it is possible (and recommended) to combine our work with for example 10-fold cross validation.

Da Costa and Schoenauer also worked on parameter tuning in the field of software testing. They developed the software environment GUIDE to help practitioners use evolutionary computation to solve hard optimization problems [130]. GUIDE contains both an easy-to-use GUI, and parameter tuning support. GUIDE has been applied to evolutionary software testing in three companies including Daimler. However, the parameter tuning offered by GUIDE is aimed for algorithms in the internal evolution engine, and not for external tools.

Biggers *et al.* highlighted that there are few studies on how to tune tools for feature location using text retrieval, and argue that it impedes deployment of such tool support [58]. They conducted a comprehensive study on the effects of different parameter settings when applying feature location using Latent Dirichlet Allocation (LDA). Their study involved feature location from six open source software systems, and they particularly discuss configurations related to indexing the source code. Biggers *et al.* report that *using default LDA settings from the literature on natural language processing is suboptimal in the context of source code retrieval*.

Thomas *et al.* addressed tuning of automated SE tools for fault localization [446]. They also emphasized the research gap considering tuning of tools, and acknowledged the challenge of finding a feasible setting for a tool using supervised learning. The paper reports from a large empirical study on 3,172 different classifier configurations, and show that *the parameter settings have a significant impact on the tool performance*. Also, Thomas *et al.* shows that ensemble learning, i.e., combining multiple classifiers, provides better performance than the best individual classifiers. However, design choices related to the combination of classifiers

also introduce additional parameter settings [254].

Lohar *et al.* discussed different configurations for SE tools supporting trace retrieval, i.e., automated creation and maintenance of trace links [313]. They propose a machine learning approach, referred to as Dynamic Trace Configuration (DTC), to search for the optimal configuration during runtime. Based on experiments with data extracted from three different domains, they show that DTC can significantly improve the accuracy of their tracing tool. Furthermore, the authors argue that DTC is easy to apply, thus supporting technology transfer. However, in contrast to TuneR, DTC is specifically targeting SE tools for trace retrieval.

ImpRec, the tool we use for the proof-of-concept evaluation of TuneR, is a type of automated SE tool that presents output as a ranked list of recommendations, analogous to well-known IR systems for web search. Modern search engines apply ranking functions that match the user and his query with web pages based on hundreds of features, e.g., location, time, search history, query content, web page title, content, and domain [489]. To combine the features in a way that yields relevant search hits among the top results, i.e., to tune the feature weighting scheme, Learning-to-Rank (LtR) is typically used in state-of-the-art web search [319]. LtR is a family of machine learning approaches to obtain feasible tuning of IR systems [312]. Unfortunately, applying LtR to the ranking function of ImpRec is not straightforward. The success of learning-to-rank in web search is enabled by enormous amounts of training data [421], manually annotated for relevance by human raters [459]. As such amounts of manually annotated training data is not available for ImpRec, and probably not for other automated SE tools either, TuneR is instead based on empirical experimentation. However, LtR is gaining attention also in SE, as showed by a recent position paper by Binkley and Lawrie [60].

4 ImpRec: An RSSE for Automated Change Impact Analysis

ImpRec is an SE tool that supports navigation among software artifacts [75], tailored for a development organization in the power and automation sector. The development context is safety-critical embedded development in the domain of industrial control systems, governed by IEC 61511 [238] and certified to a Safety Integrity Level (SIL) of 2 as defined by IEC 61508 [239]. The target system has evolved over a long time, the oldest source code was developed in the 1980s. A typical development project in the organization has a duration of 12-18 months and follows an iterative stage-gate project management model. The number of developers is in the magnitude of hundreds, distributed across sites in Europe, Asia and North America.

As specified in IEC 61511 [238], the impact of proposed software changes should be analyzed before implementation. In the case company, the impact analysis process is integrated with the issue repository. Before a corrective change is

made to resolve an issue report, the developer must store an impact analysis report as an attachment to the corresponding issue report. As part of the impact analysis, engineers are required to investigate the impact of a change, and document their findings in an impact analysis report according to a project specific template. The template is validated by an external certifying agency, and the impact analysis reports are internally reviewed and externally assessed during safety audits.

Several questions explicitly ask for trace links [71], i.e., "*a specified association between a pair of artifacts*" [203]. The engineer must specify source code that will be modified (with a file-level granularity), and also which related software artifacts need to be updated to reflect the changes, e.g., requirement specifications, design documents, test case descriptions, test scripts and user manuals. Furthermore, the impact analysis should specify which high-level system requirements cover the involved features, and which test cases should be executed to verify that the changes are correct, once implemented in the system. In the target software system, the extensive evolution has created a complex dependency web of software artifacts, thus the impact analysis is a daunting work task.

ImpRec is an RSSE that enables reuse of knowledge captured from previous impact analyses [71]. Using history mining in the issue repository, a collaboratively created trace link network is established, referred to as the knowledge base. ImpRec then calculates the centrality measure of each artifact in the knowledge base. When a developer requests impact recommendations for an issue report, ImpRec combines IR and network analysis to identify *candidate impact*. First, Apache Lucene [335] is used to search for issue reports in the issue repository that are textually similar. Then, originating from the most similar issue reports, trace links are followed both to related issue reports and to artifacts that were previously reported as impacted. Each starting point results in a set of candidate impact (set_i). When all sets of candidate impact have been established, the individual artifacts are given a weight according to a ranking function. Finally, the recommendations are presented in a ranked list in the ImpRec GUI. For further details on ImpRec, we refer to our previous publications [75, 80].

This paper presents our efforts to tune four ImpRec parameters, two related to candidate impact identification, and two dealing with ranking of the candidate impact. Figure 2 presents an overview of how ImpRec identifies candidate impact, and introduces the parameters $START$ and $LEVEL$. By setting the two parameters to high values, ImpRec identifies a large set of candidate impact. To avoid overwhelming the user with irrelevant recommendations, the artifacts in the set are ranked. As multiple starting points are used, the same artifact might be identified as potentially impacted several times, i.e., an artifact can appear in several impact sets. Consequently, the final ranking value of an individual artifact (ART_x) is calculated by summarizing how each set_i contributes to the ranking value:

$$Weight(ART_x) = \sum_{ART_x \in set_i} \frac{ALPHA * cent_x + (1 - ALPHA) * sim_x}{1 + links * PENALTY} \quad (1)$$

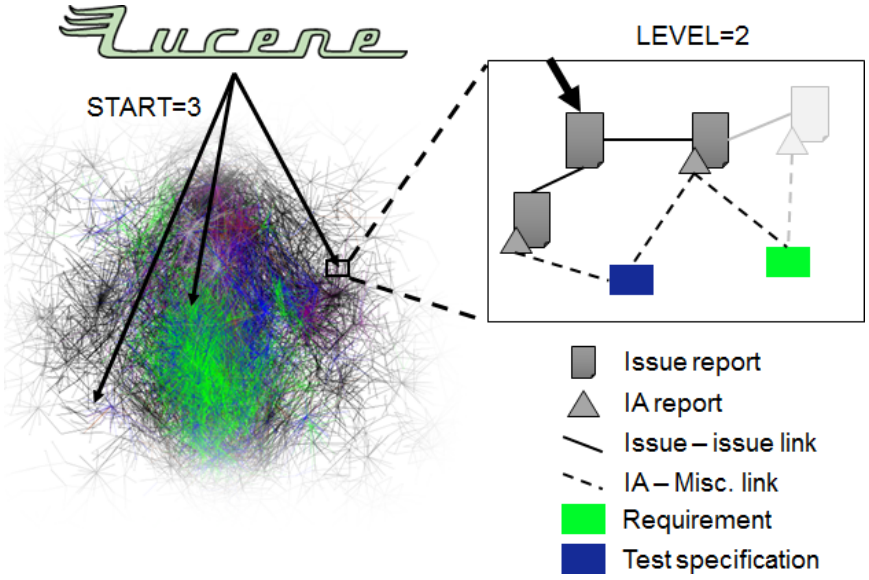


Figure 2: Identification of candidate impact using ImpRec. Two related parameters (with an example setting) are targeted for tuning: 1) The number of starting points identified using Apache Lucene ($START$), and 2) the maximum number of issue-issue links followed to identify impacted artifacts ($LEVEL$).

where $PENALTY$ is used to penalize distant artifacts and $ALPHA$ is used to set the relative importance of textual similarity and the centrality measure. sim_x is the similarity score of the corresponding starting point provided by Apache Lucene, $cent_x$ is the centrality measure of art_x in the knowledge base, and $links$ is the number of issue-issue links followed to identify the artifact (no more than $LEVEL - 1$). The rest of this paper presents TuneR, and how we used it to tune $START$, $LEVEL$, $PENALTY$, and $ALPHA$.

5 TuneR: An Experiment Framework and a Hands-on Example

This section describes the three phases of TuneR, covering 11 steps. For each step in our framework, we first describe TuneR in general terms, and then we present a hands-on example of how we tuned ImpRec. Figure 3 shows an overview of the steps in TuneR.

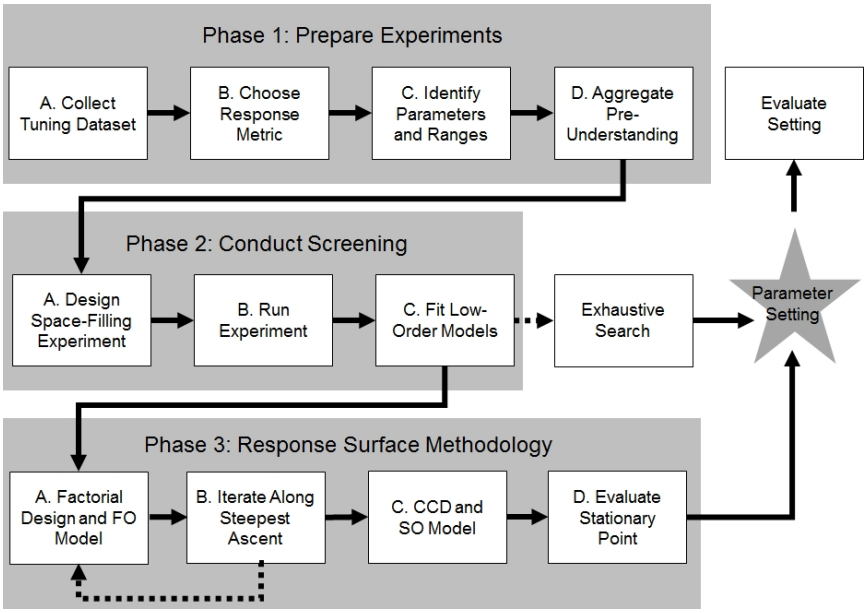


Figure 3: Overview of TuneR. The three phases are depicted in gray boxes. Dotted arrows show optional paths.

5.1 Phase 1: Prepare Experiments

Successful experimentation relies on careful planning. The first phase of TuneR consists of four steps: A) Collect Tuning Dataset, B) Choose Response Metric, C) Identify Parameters and Ranges, and D) Aggregate Pre-Understanding. All four steps are prerequisites for the subsequent Screening phase.

A) Collect Tuning Dataset

Before any tuning can commence, a dataset that properly represents the target environment must be collected. The *content validity* of the dataset refers to the representativeness of the sample in relation to all data in the target environment [457]. Thus, to ensure high content validity in tuning experiments, the experimenter must carefully select the dataset, and possibly also sample from it appropriately, as discussed by Seiffert *et al.* [423]. Important decisions that have to be made at this stage include *how old data can be considered valid* and *whether the data should be preprocessed* in any way. While a complete discussion on data collection is beyond the scope of TuneR, we capture some of the many discussions on how SE datasets should be sampled and preprocessed in this section.

In many software development projects, the characteristics of both the system under development [337] and the development process itself [320] vary considerably. If the SE tool is intended for such a dynamic target context, then it is important that the dataset does not contain obsolete data. For example, Shepperd *et al.* discuss the dangers of using old data when estimating effort in software development, and the difficulties in knowing when data turns obsolete [426]. Jonsson *et al.* show the practical significance on time locality in automated issue assignment [254], i.e., how quickly the prediction accuracy deteriorates with old training data for some projects.

Preprocessing operations, such as data filtering, influence the performance of SE tools. Menzies *et al.* even warn that variation in preprocessing steps might be a major cause of conclusion instability when evaluating SE tools [339]. Shepperd *et al.* discuss some considerations related to previous work on publicly available NASA datasets, and conclude that the importance of preprocessing in general has not been acknowledged enough [427]. Regarding filtering of datasets, Lamkanfi and Demeyer show how filtering outliers can improve prediction of issue resolution times [289], a finding that has also been confirmed by AbdelMoez *et al.* [2]. Thus, if the SE tools will be applied to filtered data, then the dataset used for the tuning experiment should be filtered as well. Another threat to experimentation with tools implementing machine learning is the *dataset shift* problem, i.e., the distribution of data in the training set differs from the test set. Turhan discuss how dataset shift relate to conclusion instability in software engineering prediction models, and presents strategies to alleviate it [454].

The tuning dataset does not only need to contain valid data, it also needs to contain enough of it. A recurring approach in SE is to evaluate tools on surrogate data, e.g., studying OSS development and extrapolating findings to proprietary contexts. Sometimes it is a valid approach, as Robinson and Francis have shown in a comparative study of 24 OSS systems and 21 proprietary software systems [404]. They conclude that the variation *among* the two categories is as big as *between* them, and, at least for certain software metrics, that there often exist OSS systems with characteristics that match proprietary systems. Several SE experiments use students as subjects, and Höst *et al.* show that it is a feasible approach under certain circumstances [225]. However, the validity of experimenting on data collected from student projects is less clear, as discussed in our previous survey [79]. Another option is to combine data from various sources, i.e., complementing proprietary data from different contexts. Tsunoda and Ono recently highlighted some risks of this approach, based on a cross-company software maintenance dataset as an example [453]. They performed a statistical analysis of the dataset, and exemplified how easy it is to detect spurious relationships between totally independent data.

As ImpRec was developed in close collaboration with industry, and is a tool tailored for a specific context, the data used for tuning must originate from the same environment. We extracted all issue reports from the issue repository, repre-

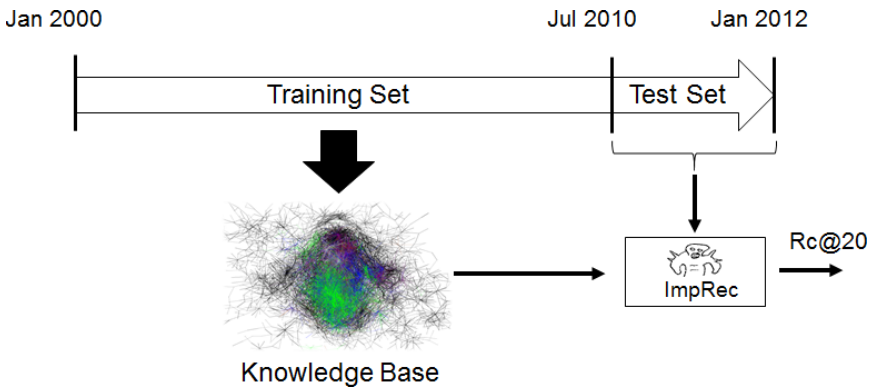


Figure 4: Composition of the ImpRec tuning dataset into training and test sets. The knowledge base is established using issue reports from Jan 2000 to Jul 2010. The subsequent issue reports are used to simulate the ImpRec response, measured in Rc@20.

senting 12 years of software evolution in the target organization [75]. As the issue reports are not independent, the internal order must be kept and we cannot use an experimental design based on cross-validation. Thus, as standard practice in machine learning evaluation, and emphasized by Arcuri and Fraser [25], we split the ordered data into non-overlapping training and test sets, as presented in Figure 4. The training set was used to establish the knowledge base, and the test set was used to measure the ImpRec performance. The experimental design used to tune ImpRec is an example of *simulation* as presented by Walker and Holmes [466], i.e., we simulate the historical inflow of issue reports to measure the ImpRec response. Before commencing the tuning experiments, we analyzed whether the content of the issue reports had changed significantly over time. Also, we discussed the evolution of both the software under development, and the development processes, with engineers in the organization. We concluded that we could use the full dataset for our experiments, and we chose not to filter the dataset in any way.

B) Choose Response Metric

The next step in TuneR is to choose what metric to base the tuning on. TuneR is used to optimize a response with regard to a single metric, as it relies on traditional RSM, thus the response metric needs to be chosen carefully. Despite mature guidelines like the *Goal-Question-Metric* framework [41], the dangers of software measurements have been emphasized by several researchers [88, 156, 262]. However, we argue that selecting a metric for the response of an SE tool is a far more reasonable task than measuring the entire software development process based on a single metric. A developer of an SE tool probably already knows the precise

goal of the tool, and thus should be able to choose or invent a feasible metric. Moreover, if more than one metric is important to the response, the experimenter can introduce a compound metric, i.e., a combination of individual metrics. On the other hand, no matter what metric is selected, there is a risk that naïvely tuning with regard to the specific metric leads to a sub-optimal outcome, a threat further discussed in Section 5.4.

Regarding the tuning of ImpRec, we rely on the comprehensive research available on quantitative IR evaluation, e.g., the TREC conference series and the Cranfield experiments [465]. In line with general purpose search systems, ImpRec presents a ranked list of candidates for the user to consider. Consequently, it is convenient to measure the quality of the output using established IR measures for ranked retrieval. The most common way to evaluate the effectiveness of an IR system is to measure precision and recall. Precision is the fraction of retrieved documents that are relevant, while recall is the fraction of relevant documents that are retrieved. As there is a trade-off between precision and recall, they are often reported pairwise. The pairs are typically considered at fixed recall levels (e.g., 0.1...1.0), or at specific cut-offs of the ranked list (e.g., the top 5, 10, or 20 items) [325].

We assume that a developer is unlikely to browse too many recommendations from ImpRec. Consequently, we use a cut-off point of 20 to disregard all recommendations below that rank. While it is twice as many as the standardized page-worth output from search engines, CIA is a challenging task in which practitioners request additional tool support [80, 136], and thus we assume that engineers are willing to browse additional search hits. Also, we think that engineers can quickly filter out the interesting recommendations among the top 20 hits.

Several other measures for evaluating the performance of IR systems have been defined. A frequent compound measure is the F-score, a harmonized mean of precision and recall. Other more sophisticated metrics include Mean Average Precision (MAP) and Normalized Discounted Cumulative Gain (NDCG) [325]. However, for the tuning experiments in this paper, we decide to optimize the response with regard to *recall considering the top-20 results* (Rc@20).

C) Identify Parameters and Specify Ranges for Normal Operation

The third step of Phase 1 in TuneR concerns identification of parameters to vary during the tuning experiments. While some parameters might be obvious, maybe as explicit in settings dialogs or configuration files, other parameters can be harder to identify. Important variation points may be hidden in the implementation of the SE tool, thus identifying what actually constitutes a meaningful parameter can be challenging.

Once the parameters have been identified, the experimenter needs to decide what levels should be used. A first step is, in line with standard DoE practice [165, pp. 213], to identify what range represents "normal operation" for each parameter.

Table 1: The four parameters studied in the tuning experiment, and the values that represent their range for normal operation.

Parameter	Type of range	Normal range
$ALPHA$	Non-negative bounded continuous	[0-1]
$PENALTY$	Non-negative continuous	[0-5]
$START$	Positive discrete	[1-200]
$LEVEL$	Positive discrete	[1-10]

Parameter variations within such a range should be large enough to cause changes in the response, but the range should not cover so distant values that the fundamental characteristics of the tool are altered. For some parameters, identification of the normal range is straightforward because of well-defined bounds, e.g., a real value between 0 and 1 or positive integers between 1 and 10. For other parameters, however, it is possible that neither the bounds nor even the sign is known. Parameters can also be binary or categorical, taking discrete values [165, pp. 208].

Regarding ImpRec, Section 4 already presented the four parameters $ALPHA$, $PENALTY$, $START$, and $LEVEL$. However, also the search engine library Apache Lucene is highly configurable. But as configuring Lucene is complex, see for example McCandless *et al.* [335, Ch. 2], and since the default setting yielded useful results in our previous work on issue reports [78], we choose to consider it as a black box with fixed parameters in this study, i.e., we use the default setting. The other four parameters of ImpRec on the other hand, do not have any default values, thus we must continue by specifying ranges for normal operation.

Table 1 shows how we specify the ranges for normal operation for the four parameters. $ALPHA$ represents the relative importance between textual similarities and centrality measures, i.e., it is a bounded real value between 0 and 1, and we consider the full range normal. $START$ is a positive integer, there must be at least one starting point, but there is no strict upper limit. We consider 200 to be the upper limit under normal operation, as we suspect larger values to generate imprecise recommendations and too long response times. $LEVEL$ and $PENALTY$ both deal with following links between issue reports in the knowledge base. Analogous to the argumentation regarding $START$, we suspect that assigning $LEVEL$ a too high value might be counter-productive. $LEVEL$ must be a positive integer, as 1 represents not following any issue-issue links at all. We decide to consider [1, 10] as the range for $LEVEL$ under normal operation. $PENALTY$ downweighs potential impact that has been identified several steps away in the knowledge base, i.e., impact with a high level. The parameter can be set to any non-negative number, but we assume that a value between 0 and 5 represents normal operation. Already the level 5 would make the contribution of distant issue reports practically zero, see Equation 1.

D) Aggregate Pre-Understanding

Successful tuning of an SE tool requires deep knowledge. The experimenter will inevitably learn about the tool in the next two phases of TuneR, but probably there are already insights before the experimentation commences. In line with Gummesson's view [209, pp. 75], we value this *pre-understanding* as fundamental to reach deep understanding. The pre-understanding can provide the experimenter with a shortcut to a feasible setting, as it might suggest in what region the optimal setting is located. To emphasize this potential, TuneR consists of a separate step aimed at recapitulating what has already been experienced.

The development of ImpRec was inspired by test-driven development [46], thus we tried numerous different parameter settings in test cases during development. By exploring different settings in our trial runs during development, an initial parameter tuning evolved as a by-product of the tool development. While we performed this experimentation in an *ad hoc* fashion, we measured the output with regard to Rc@20, and recorded the results in a structured manner. Recapitulating our pre-understanding regarding the parameters provides the possibility to later validate the outcome of the screening in Phase 2 of TuneR.

The *ad hoc* experiments during development contain results from about 100 trial runs. We explored $ALPHA$ ranging from 0.1 to 0.9, obtaining the best results for high values. START had been varied between 3 and 20, and again high values appeared to be a better choice. Finally, we had explored LEVEL between 3 and 10, and PENALTY between 0 and 8. Using a high LEVEL and low PENALTY yielded the best results. Based on our experiences, we deployed ImpRec in the organization using the following default setting: $ALPHA = 0.83, START = 17, LEVEL = 7, PENALTY = 0.2$ (discussed in depth in another paper [80]). The default setting yields a response of Rc@20=0.41875, i.e., about 40% of the true impact is delivered among the top-20 recommendations. We summarize our expectations as follows:

- The ranking function should give higher weights to centrality measures than textual similarity ($0.75 < ALPHA < 1$)
- Many starting points benefit the identification of impact ($START > 15$)
- Following related cases several steps away from the starting point improves results ($LEVEL > 5$)
- We expect an interaction between $LEVEL$ and $PENALTY$, i.e., that increasing the number of levels to follow would make penalizing distant artifacts more important
- Completing an experimental run takes about 10-30 s, depending mostly on the value of $START$.

5.2 Phase 2: Conduct Screening Experiment

Phase 2 in TuneR constitutes three steps related to screening. Screening experiments are conducted to identify the most important parameters in a specific context [347, pp. 6] [165, pp. 240]. Traditional DoE uses 2^k factorial design for screening, using broad values (i.e., high and low values within the range of normal operation) to calculate main effects and interaction effects. However, as explained in Section 2.3, space-filling design should be applied when tuning SE tools. The three screening steps in TuneR are: A) Design Space-Filling Experiment, B) Run Experiment, and C) Fit Low-Order Models. Phase 2 concludes by identifying a *promising region*, i.e., a setting that appears to yield a good response, a region that is used as input to Phase 3.

A) Design a Space-Filling Experiment

The first step in Phase 2 in TuneR deals with designing a space-filling screening experiment. The intention of the screening is not to fully analyze how the parameters affect the response, but to complement the less formal pre-understanding. Still, the screening experiment will consist of multiple runs. As a rule of thumb, Levy and Steinberg approximate that the number of experimental runs needed in a DoCE screening is ten times the number of parameters involved [299].

Several aspects influence the details of the space-filling design, and we discuss four considerations below. First, parameters of different types (as discussed in Phase 1, Step B) require different experimental settings. The space of categorical parameters can only be explored by trying all levels. Bounded parameters on the other hand can be explored using uniform space-filling designs as presented in Section 2.2. Unbounded parameters however, at least when the range of normal operation is unknown, requires the experimenter to select values using other approaches. Second, our pre-understanding from Phase 1, Step D might suggest that some parameters are worth to study using more fine-granular values than others. In such cases, the pre-understanding has already contributed with a preliminary sensitivity analysis [418, pp. 189], and the design should be adjusted accordingly. Third, the time needed to perform the experiments limits the number of experimental runs, in line with discussions on search budget in SBSE [213]. Certain parameter settings might require longer execution times than others, and thus require a disproportional amount of the search budget. Fourth, there might be known constraints at play, forcing the experimenter to avoid certain parameter values. This phenomenon is in line with the discussion on unsafe settings in DoE [165, pp. 254].

Unless the considerations above suggest special treatment, we propose the following rules-of-thumb as a starting point:

- Restrict the search budget for the screening experiment to a maximum of 48 h, i.e., it should not require more than a weekend to execute.

Table 2: Screening design for the four parameters $ALPHA$, $PENALTY$, $START$, and $LEVEL$.

Parameter	#Levels	Values
$ALPHA$	7	0.01, 0.17, 0.33, 0.5, 0.67, 0.83, 0.99
$PENALTY$	7	0.01, 0.1, 0.5, 1, 2, 4, 8
$START$	10	1, 2, 4, 8, 16, 32, 64, 128, 256, 512
$LEVEL$	7	1, 2, 4, 8, 16, 32, 64

- Use the search budget to explore the parameters evenly, i.e., for an SE tool with i parameters, and the search budget allows n experimental runs, use $\sqrt[i]{n}$ values for each parameter.

- Apply a uniform design for bounded parameters, i.e., spread the parameter values evenly.

- Use a geometric series of values for unbounded parameters, e.g., for integer parameters explore values $2^i,\ i = 0,\ 1,\ 2,\ 3,\ 4$ ...

When screening the parameters of ImpRec, we want to finish the experimental runs between two workdays (4 PM to 8 AM, 16 h) to enable an analysis of the results on the second day. Based on our pre-understanding, we predict that on average four experimental runs can be completed per minute, thus about 3,840 experimental runs can be completed within the 16 h search budget. As we have four parameters, we can evaluate about $\sqrt[4]{3,840} \approx 7.9$ values per parameter, i.e., 7 rounded down.

Table 2 shows the values we choose for screening the parameters of ImpRec. $ALPHA$ is a relative weighting parameter between 0 and 1. We use a uniform design to screen $ALPHA$, but do not pick the boundary values to avoid divisions by zero. $PENALTY$ is a positive continuous variable with no upper limit, and we decide to evaluate several magnitudes of values. A penalty of 8 means that the contribution of distant artifacts to the ranking function is close to zero, thus we do not need to try higher values. $START$ and $LEVEL$ are both positive discrete parameters, both dealing with how many impact candidates should be considered by the ranking function. Furthermore, our pre-understanding shows that the running time is proportional to the value of $START$. As we do not know how high values of $START$ are feasible, we choose to evaluate up to 512, a value that represents about 10% of the full dataset. Exploring such high values for $LEVEL$ does not make sense, as there are no such long chains of issue reports. Consequently, we limit $LEVEL$ to 64, already a high number. In total, this experimental design, constituting 3,430 runs, appears to be within the available search budget.

B) Run Screening Experiment

When the design of the screening experiment is ready, the next step is to run the experiment. To enable execution of thousands of experimental runs, a stable experiment framework for automatic execution must be developed. Several workbenches are available that enable reproducible experiments, e.g., frameworks such as Weka [185] and RapidMiner [222] for general purpose machine learning and data mining, and SE specific efforts such as the TraceLab workbench [266] for traceability experiments, and the more general experimental Software Engineering Environment (eSSE) [452]. Furthermore, the results should be automatically documented as the experimental runs are completed, in a structured format that supports subsequent analysis.

We implement a feature in an experimental version of ImpRec that allows us to execute a sequence of experimental runs. Also, we implement an evaluation feature that compares the ImpRec output to a 'gold standard' (see the 'static validation' in our parallel publication [80] for a detailed description), and calculates established IR measures, e.g., precision, recall, and MAP at different cut-off levels. Finally, we print the results of each experimental run as a separate row in a file of Comma Separated Values (CSV). Listing V.1 shows an excerpt of the resulting csv-file, generated from our screening experiment. The first four columns show the parameter values, and the final column is the response measured in Rc@20.

Listing V.1: screening.csv generated from the ImpRec screening experiment.

```
alpha, penalty, start, level, resp
0.01, 0.01, 1, 1, 0.059375
0.01, 0.01, 1, 2, 0.078125
0.01, 0.01, 1, 4, 0.1125
0.01, 0.01, 1, 8, 0.115625
0.01, 0.01, 1, 16, 0.115625
...
(3,420 additional rows)
...
0.99, 8, 512, 4, 0.346875
0.99, 8, 512, 8, 0.315625
0.99, 8, 512, 16, 0.31875
0.99, 8, 512, 32, 0.321875
0.99, 8, 512, 64, 0.328125
```

C) Fit Low-order Polynomial Models

The final step in Phase 2 of TuneR involves analyzing the results from the screening experiment. A recurring observation in DoE is that only a few factors dominate the response, giving rise to well-known principles such as the '80-20 rule' and 'Occam's razor' [277, pp. 157]. In this step, the goal is to find the simplest polynomial model that can be used to explain the observed response. If neither a first nor second-order polynomial model (i.e., linear and quadratic effects plus two-way

interactions) fits the observations from the screening experiment, the response surface is complex. Modelling a complex response surfaces is beyond the scope of TuneR, as it requires advanced techniques such as neural networks [347, pp. 446], splines, or kriging [251]. If low-order polynomial models do not fit the response, TuneR instead relies on quasi-exhaustive space-filling designs (see Fig. 3). We discuss this further in Section 6, where we use exhaustive search to validate the result of the ImpRec tuning using TuneR.

When a low-order polynomial model has been fit, it might be possible to simplify it by removing parameters that do not influence the response much. The idea is that removal of irrelevant and noisy variables should improve the model. Note, however, that this process known as subset selection in linear regression, has been widely debated among statisticians, referred to as "fishing expeditions" and other derogatory terms (see for example discussions by Lukacs *et al.* [318] and Miller [340, pp. 8]). Still, when tuning an SE tool with a multitude of parameters, reducing the number of factors might be a necessary step for computational reasons. Moreover, working with a reduced set of parameters might reduce the risk of overfitting [14]. A standard approach is stepwise backward elimination [395, pp. 336], i.e., to iteratively remove parameters until all that remain have a significant effect on the response. While parameters with high p-values are candidates for removal [445, pp. 277], all such operations should be done with careful consideration. We recommend visualizing the data (cf. Fig. 5 and 6), and trying to understand why the screening experiment resulted in the response. Also, note that any parameter involved in interaction or quadratic effects must be kept.

To fit low-order polynomial models for ImpRec's response surface, we use the R package *rsm* [294], and the package *visreg* [84] to visualize the results. Assuming that screening.csv has been loaded to screening, Listing V.2 and V.3 fit a first-order and second-order polynomial model, respectively.

Listing V.2: Fitting a first-order polynomial model with *rsm* [294]. The results are truncated.

```
> FO_model <- rsm(resp ~ FO(alpha, penalty, start, level), data=screening)
> summary(FO_model)

Call:
rsm(formula = resp ~ FO(alpha, penalty, start, level), data = screening)

              Estimate Std. Error t value Pr(>|t|)
(Intercept) 2.4976e-01 4.2850e-03 58.2855   <2e-16 ***
alpha       4.9432e-02 5.7393e-03  8.6129   <2e-16 ***
penalty     8.8721e-04 7.0248e-04  1.2630   0.2067
start       1.2453e-04 1.2052e-05 10.3327   <2e-16 ***
level       6.9603e-05 8.8805e-05  0.7838   0.4332
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Multiple R-squared:  0.05076,    Adjusted R-squared:  0.04965
F-statistic: 45.79 on 4 and 3425 DF,  p-value: < 2.2e-16
```

```
18
19 Analysis of Variance Table
20
21 Response: resp
22                                    Df Sum Sq Mean Sq F value    Pr(>F)
23 FO(alpha, penalty, start, level)    4  2.234 0.55859  45.789 < 2.2e-16
24 Residuals                        3425 41.782 0.01220
25 Lack of fit                      3425 41.782 0.01220
26 Pure error                          0  0.000
```

The second order model fits the response better than the first order model; the lack of fit sum of squares is 29.1841 versus 41.782 (cf. Listing V.3:62 and Listing V.2:25). Moreover, Listing V.3:44-47 show that the parameters $PENALTY$, $START$, and $LEVEL$ have a quadratic effect on the response. Also, interaction effects are significant, as shown by alpha:start, penalty:start, and start:level (cf. Listing V.3:38-43). Figure 5 visualizes[3] how the second order model fits the response, divided into the four parameters. As each data point represents an experimental run, we conclude that there is a large spread in the response. For most individual parameter values, there are experimental runs that yield an Rc@20 between approximately 0.1 and 0.4. Also, in line with Listing V.3, we see that increasing $START$ appears to improve the response, but the second order model does not fit particularly well.

Listing V.3: Fitting a second-order polynomial model with *rsm* [294]. The results are truncated.

```
27 > SO_model <- rsm(resp ~ SO(alpha, penalty, start, level), data=screening)
28 > summary(SO_model)
29 Call:
30 rsm(formula = resp ~ SO(alpha, penalty, start, level), data = screening)
31
32                   Estimate  Std. Error  t value  Pr(>|t|)
33 (Intercept)     2.1502e-01  6.1700e-03  34.8493 < 2.2e-16 ***
34 alpha           2.6868e-02  1.8997e-02   1.4143 0.1573604
35 penalty         4.1253e-03  2.4574e-03   1.6787 0.0932935 .
36 start           1.2814e-03  4.1704e-05  30.7247 < 2.2e-16 ***
37 level           1.2045e-03  3.2053e-04   3.7579 0.0001742 ***
38 alpha:penalty  -4.5460e-04  1.7894e-03  -0.2541 0.7994640
39 alpha:start     3.3458e-04  3.0698e-05  10.8993 < 2.2e-16 ***
40 alpha:level     5.5608e-05  2.2620e-04   0.2458 0.8058257
41 penalty:start   3.3783e-06  3.7573e-06   0.8991 0.3686588
42 penalty:level   6.7390e-05  2.7687e-05   2.4340 0.0149839 *
43 start:level    -4.9485e-06  4.7499e-07 -10.4182 < 2.2e-16 ***
44 alpha^2        -1.1659e-02  1.7181e-02  -0.6786 0.4974522
45 penalty^2      -5.8485e-04  2.7071e-04  -2.1604 0.0308128 *
46 start^2        -2.5851e-06  7.3816e-08 -35.0212 < 2.2e-16 ***
47 level^2        -1.2702e-05  4.4041e-06  -2.8840 0.0039508 **
48 ---
49 Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
50
```

[3] R command: $> visreg(SO\_model)$

```
51  Multiple R-squared:  0.337,     Adjusted R-squared:  0.3342
52  F-statistic:  124 on 14 and 3415 DF,  p-value: < 2.2e-16
53
54  Analysis of Variance Table
55
56  Response: resp
57                                    Df  Sum Sq Mean Sq F value    Pr(>F)
58  FO(alpha, penalty, start, level)   4  2.2343 0.55859  65.363 < 2.2e-16
59  TWI(alpha, penalty, start, level)  6  2.0014 0.33356  39.032 < 2.2e-16
60  PQ(alpha, penalty, start, level)   4 10.5963 2.64907 309.983 < 2.2e-16
61  Residuals                       3415 29.1841 0.00855
62  Lack of fit                     3415 29.1841 0.00855
63  Pure error                         0  0.0000
```

Listing V.3 suggests that all four parameters are important when modelling the response surface. The statistical significance of the two parameters $START$ and $LEVEL$ is stronger than for $ALPHA$ and $PENALTY$. However, $ALPHA$ is involved in a highly significant interaction effect (alpha:start in Listing V.3:39). Also, the quadratic effect of $PENALTY$ on the response is significant (penalty^2 in Listing V.3:45). Consequently, we do not simplify the second order model of the ImpRec response by reducing the number of parameters.

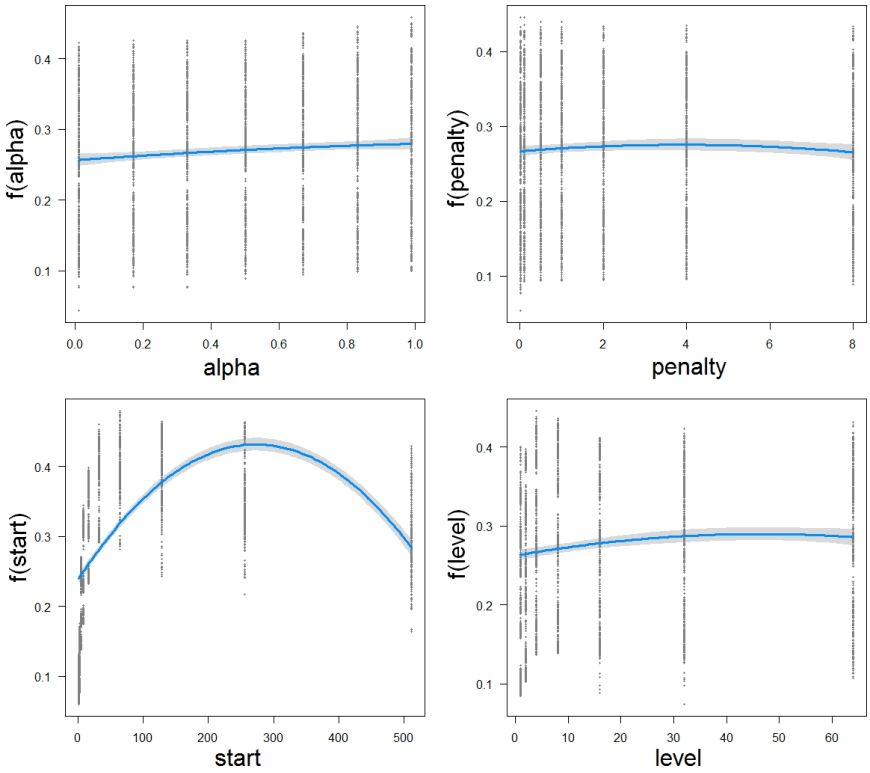


Figure 5: Visualization of the second order model using *visreg* [84].

Figure 6 displays boxplots of the response per parameter, generated with $ggplot2$[4] [470]. Based on the boxplots, we decide that a promising region for further tuning appears to involve high $ALPHA$ values, $START$ between 32 and 128, and $LEVEL = 4$. The parameter value of $PENALTY$ however, does not matter much, as long as it is not too small, thus we consider values around 1 promising. An experimental run with the setting $ALPHA = 0.9, PENALTY = 1, START = 64, LEVEL = 4$ gives a response of Rc@20=0.46875, compared to 0.41875 for the default setting. Thus, this 11.9% increase of the response confirms the choice of a promising region.

We summarize the results from screening the ImpRec parameters as follows:

- Centrality values of artifacts are more important than textual similarity when predicting impact ($ALPHA$ close to 1). Thus, previously impacted artifacts

[4] R commands for the $START$ parameter:
$> start\_box <- ggplot(screening, aes(factor(start), resp))$
$> start\_box + geom\_boxplot()$

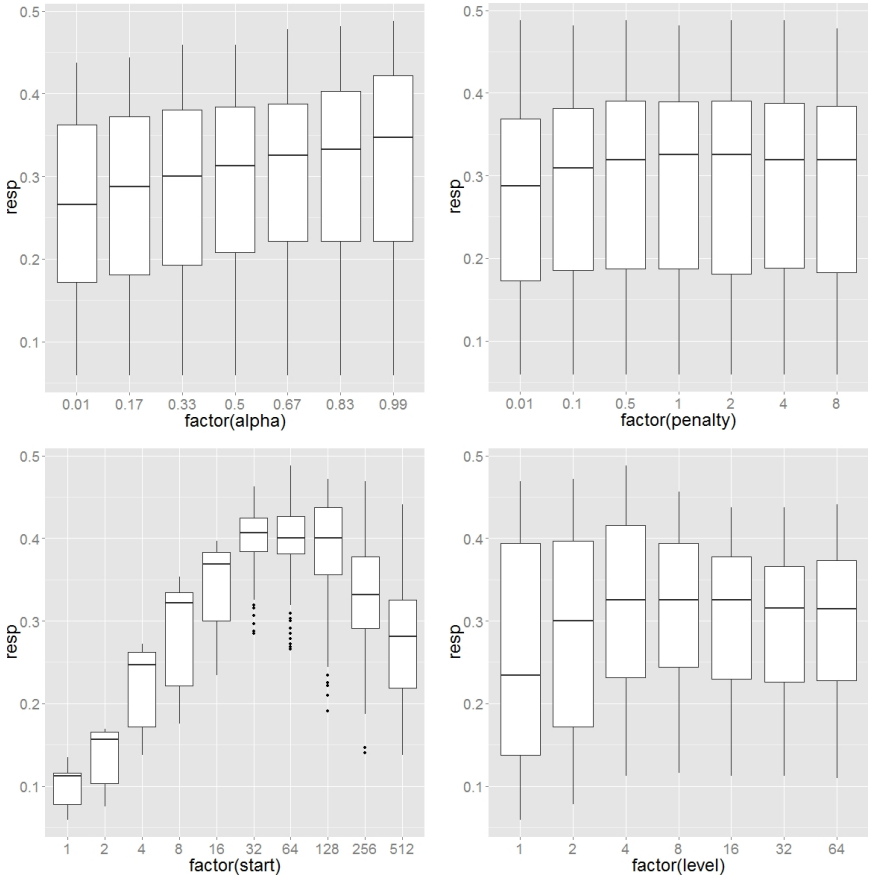


Figure 6: Value of the response for different parameter settings. Note that the x-axis is only linear in the first plot ($ALPHA$).

(i.e., artifacts with high centrality in the network) are likely to be impacted again.

- The low accuracy of the textual similarity is also reflected by the high parameter value of $START$; many starting points should be used as compensation.
- Regarding $LEVEL$ and $PENALTY$ we observe that following a handful of issue-issue links is beneficial, trying even broader searches however is not worthwhile.
- Also, severely penalizing distant artifacts does not benefit the approach, i.e., most related issues are meaningful to consider.

- A promising region, i.e., a suitable start setting for Phase 3, appears to be around $ALPHA = 0.9, PENALTY = 1, START = 64, LEVEL = 4$.

5.3 Phase 3: Apply Response Surface Methodology

The third phase in TuneR uses RSM to identify the optimal setting. The first part of RSM is an iterative process. We use a factorial design to fit a first-order model to the response surface, and then gradually modify the settings along the most promising direction, i.e., the *path of the steepest ascent*. Then, once further changes along that path do not yield improved responses, the intention is to pin-point the optimal setting in the vicinity. The pin-pointing is based on analyzing the stationary points of a second-order fit of that particular region of the response surface, determined by applying an experiment using CCD (cf. Fig. 1). We describe Step A and B (i.e., the iterative part) together in the following subsection, and present Step C and D in the subsequent subsections.

When applying RSM, an important aspect is to use an appropriate *coding transformation*. The way the data are coded affects the results of the steepest ascent analysis. If all coded variables in the experiment vary over the same range, typically -1 and 1, each parameter gets an equal share in potentially determining the steepest ascent path [294].

A and B: Factorial designs, First-order Models, and Iteration

Iteration of the first two steps is intended to quickly move toward the optimum. To find the direction, we design an experiment using 2^k factorial design and fit a first-order model of the response surface. The factorial design uses the outcome from Phase 2 as the center point, and for each parameter, we select a high value and a low value, referred to as the *factorial range* [165]. Selecting a feasible factorial range is one of the major challenges in RSM, another one is to select an appropriate *step size*.

Selecting a suitable factorial range for a computer experiment is a bit different than for a physical experiment. In traditional DoE, a too narrow range generates a factorial experiment dominated by noise. While noise is not a threat in experiments aimed at tuning SE tools, a too narrow range will instead not show any difference in the response at all. On the other hand, the range can also be too broad, as the response surface might then be generalized too much. Dunn reports that selecting extreme values is a common mistake in DoE, and suggests selecting 25% of the extreme range as a rule-of-thumb [165]. Since the number of tuning experiments typically is not limited in the same way as physical experiments, it is possible to gradually increase the factorial range until there is a difference in the response.

The factorial experiment yields the direction of the steepest ascent, but the next question is how much to adjust the setting in that direction, i.e., the step size. Again we want the difference to be large enough to cause a change in the

Table 3: First RSM iteration, 2^k factorial design for the four parameters $ALPHA$, $PENALTY$, $START$, and $LEVEL$.

	Coded variables				Natural variables				
Exp. Run	x1	x2	x3	x4	ALPHA	PENALTY	START	LEVEL	Resp.
1	-1	-1	-1	-1	0.85	0.8	60	3	0.468750
2	1	-1	-1	-1	0.95	0.8	60	3	0.481250
3	-1	1	-1	-1	0.85	1.2	60	3	0.468750
4	1	1	-1	-1	0.95	1.2	60	3	0.478125
5	-1	-1	1	-1	0.85	0.8	68	3	0.478125
6	1	-1	1	-1	0.95	0.8	68	3	0.484375
7	-1	1	1	-1	0.85	1.2	68	3	0.475000
8	1	1	1	-1	0.95	1.2	68	3	0.484375
9	-1	-1	-1	1	0.85	0.8	60	5	0.471875
10	1	-1	-1	1	0.95	0.8	60	5	0.478125
11	-1	1	-1	1	0.85	1.2	60	5	0.471875
12	1	1	-1	1	0.95	1.2	60	5	0.478125
13	-1	-1	1	1	0.85	0.8	68	5	0.468750
14	1	-1	1	1	0.95	0.8	68	5	0.484375
15	-1	1	1	1	0.85	1.2	68	5	0.468750
16	1	1	1	1	0.95	1.2	68	5	0.481250

response in a reasonable amount of experiments, but not so large that we move over an optimum. A good decision relies on the experimenter's understanding of the parameters involved in the SE tool. Otherwise, a rule-of-thumb is to choose a step size equal to the value of the largest coefficient describing the direction of the steepest ascent [40].

For tuning ImpRec, we decide to fit a first-order model in the region: $ALPHA = 0.9 \pm 0.05, PENALTY = 1 \pm 0.5, START = 64 \pm 4, LEVEL = 4 \pm 1$. Our experience from the screening experiments suggests that these levels should result in a measurable change in the response. Table 3 shows the 2^k factorial design we apply, and the results from the 16 experimental runs. We report the experimental runs in Yates' standard order according to the DoE convention, i.e., starting with low values, and then alternating the sign of the first variable the fastest, and the last variable the slowest [342, pp. 237]. Finally, we store the table, except the coded variables, in $rsm1\_factorial.csv$. Listing V.4 shows the analysis of the results, conducted in coded variables. The standard coding transformation from a natural variable v_N to a coded variable v_C in DoE is [165, pp. 245]:

$$v_c = \frac{v_n - center_v}{\Delta_v/2} \tag{2}$$

where Δ_v is the factorial range of v_n, and $center_v$ is its center point. For the four parameters of ImpRec, the coding is presented in Listing V.4 on line 2.

Listing V.4 reveals that $x1$ and $x3$ (i.e., $ALPHA$ and $START$ in coded values) affect the response the most. As visualizing the response surface in more than two variables is difficult, Figure 7 shows the contour plot[5] wrt. $x1$ and $x3$, generated using *visreg* [84]. Our experiments suggest that higher responses can be achieved if we increase $ALPHA$ and $START$, and decrease $PENALTY$ and $LEVEL$. We decide to use the step size provided by the direction of the steepest ascent in original units, as it already constitutes actionable changes to the parameters (cf. Listing V.4:95). Table 4 shows the experimental results when gradually changing the ImpRec settings in the direction: $(+0.046, -0.0223, +1.338, -0.111)$. Note that $START$ and $LEVEL$ are integer parameters and thus rounded off accordingly (highlighted in italic font), and that $ALPHA$ has a maximum value of 1 (or 0.99 for practical reasons). We observe that the response continuously improves until step 10 (in bold font in Table 4). Two additional steps in the same direction confirm the decreased response.

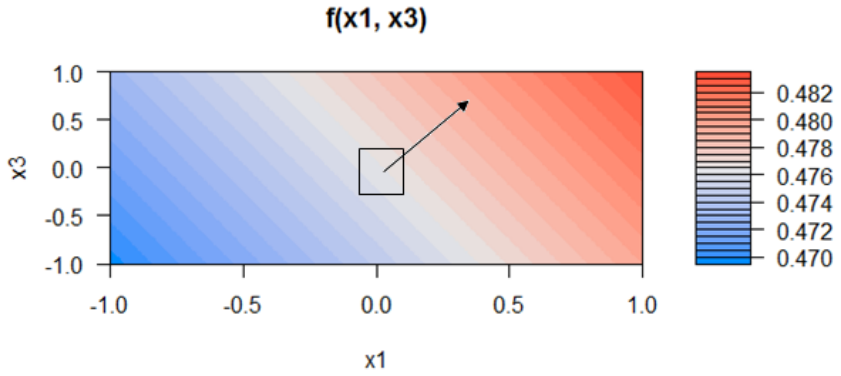


Figure 7: Contour plot displaying the two most important parameters ($ALPHA$ and $START$) in coded variables, generated using *visreg* [84]. We have added an arrow showing the direction of the steepest ascent.

Listing V.4: Using *rsm* [294] to find the direction of the steepest ascent.

```
64 > rsm1 <- read.csv("rsm1_factorial.csv")
65 > rsm1_coded <- coded.data(rsm1, x1~(alpha -0.9)/0.05,
66 x2~(penalty -1)/0.2, x3~(start -64)/4, x4~(level -4)/1)
67 > rsm1_model <- rsm(resp~FO(x1, x2, x3, x4), data=rsm1_coded)
68 > summary(rsm1_model)
69
70 Call:
71 rsm(formula = resp ~ FO(x1, x2, x3, x4), data = rsm1_coded)
72
73                Estimate   Std. Error  t value  Pr(>|t|)
74 (Intercept)  0.47636719  0.00069429 686.1210 < 2.2e-16 ***
```

[5] R command: $> visreg(rsm1\_model, "x1", "x3")$

Table 4: Iterative exploration along the direction of the steepest ascent. The tenth step, presented in bold font, results in a decreased response. Values in italic font are rounded off to the nearest integer value.

Step	*ALPHA*	*PENALTY*	*START*	*LEVEL*	Resp.
0	0.9	1	64	4	0.46875
1	0.946456	0.977701	*65.33793*	*3.888506*	0.471875
2	0.992912	0.955402	*66.67586*	*3.777012*	0.4875
3	0.99	0.933104	*68.01379*	*3.665518*	0.4875
4	0.99	0.910805	*69.35172*	*3.554024*	0.4875
5	0.99	0.888506	*70.68965*	*3.442529*	0.490625
6	0.99	0.866207	*72.02758*	*3.331035*	0.49375
7	0.99	0.843908	*73.36551*	*3.219541*	0.5
8	0.99	0.821609	*74.70344*	*3.108047*	0.50625
9	0.99	0.799311	*76.04137*	*2.996553*	0.50625
10	**0.99**	**0.777012**	**77.37929**	**2.885059**	**0.49375**
11	0.99	0.754713	*78.71722*	*2.773565*	0.490625
12	0.99	0.732414	*80.05515*	*2.662071*	0.490625

```
x1           0.00488281  0.00069429   7.0328 2.175e-05 ***
x2          -0.00058594  0.00069429  -0.8439  0.41668
x3           0.00175781  0.00069429   2.5318  0.02788 *
x4          -0.00058594  0.00069429  -0.8439  0.41668
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Multiple R-squared:  0.8389,    Adjusted R-squared:  0.7804
F-statistic: 14.32 on 4 and 11 DF,  p-value: 0.0002442

Analysis of Variance Table

Response: resp
                Df     Sum Sq    Mean Sq F value    Pr(>F)
FO(x1, x2, x3, x4)  4 0.00044189 1.1047e-04  14.324 0.0002442
Residuals          11 0.00008484 7.7130e-06
Lack of fit        11 0.00008484 7.7130e-06
Pure error          0 0.00000000

Direction of steepest ascent (at radius 1):
        x1         x2         x3         x4
 0.9291177 -0.1114941  0.3344824 -0.1114941

Corresponding increment in original units:
      alpha     penalty       start       level
 0.04645588 -0.02229882  1.33792946 -0.11149412
```

The second iteration starts where the first ended, i.e., using the tenth step in Table 4 as its center point. The parameter $ALPHA$ is already at its maximum value,

Table 5: Second RSM iteration, 2^k factorial design for the three parameters $PENALTY$, $START$, and $LEVEL$.

	Coded variables			Natural variables			
Exp. Run	$x1$	$x2$	$x3$	$PENALTY$	$START$	$LEVEL$	Resp.
1	-1	-1	-1	0.76	76	2	0.465625
2	1	-1	-1	0.84	76	2	0.5
3	-1	1	-1	0.76	80	2	0.4625
4	1	1	-1	0.84	80	2	0.490625
5	-1	-1	1	0.76	76	4	0.465625
6	1	-1	1	0.84	76	4	0.5
7	-1	1	1	0.76	80	4	0.4625
8	1	1	1	0.84	80	4	0.490625

thus we focus on $PENALTY$, $START$, and $LEVEL$. We decide to use the following factorial ranges: $PENALTY = 0.80 \pm 0.04, START = 78 \pm 2$, and $LEVEL = 3 \pm 1$. Table 5 shows the corresponding 2^k factorial experiment. We store the table, except the coded variables, in $rsm2\_factorial.csv$. Listing V.5 shows the analysis of the results, including the coding transformation of the parameters.

Listing V.5 shows that the direction of the steepest ascent involves changing the value of $START$ and $LEVEL$, but not $PENALTY$. We also know that the setting $(0.99, 0.80, 76, 3)$ yields 0.50625 (cf., step 9 in Table 4). Table 6 shows the results from iteratively moving from this setting along the direction of the steepest ascent. As we do not observe any increases in the response when changing the two integer parameters $START$ and $LEVEL$, we conclude that this ImpRec setting is in the region of the maximum. In the next TuneR step, the goal is to pin-point the exact position of the stationary point.

Listing V.5: Second RSM iteration, using *rsm* [294] to find a new direction of the steepest ascent.

```
101 > rsm2 <- read.csv("rsm2_factorial.csv")
102 > rsm2_coded <- coded.data(rsm2, x2~(penalty -0.80)/0.04,
103 x3~(start -78)/2, x4~(level -3)/1)
104 > rsm2_model <- rsm(resp~FO(x2, x3, x4), data=rsm2_coded)
105 > summary(rsm2_model)
106
107 Call:
108 rsm(formula = resp ~ FO(x2, x3, x4), data = rsm2_coded)
109
110                Estimate  Std. Error t value  Pr(>|t|)
111 (Intercept) 4.7969e-01 7.8125e-04     614 4.222e-11 ***
112 x2          6.7465e-18 7.8125e-04       0   1.00000
113 x3         -3.1250e-03 7.8125e-04      -4   0.01613 *
114 x4          1.5625e-02 7.8125e-04      20 3.688e-05 ***
115 ---
```

Table 6: Iterative exploration along the new direction of the steepest ascent. No changes result in an increased response, indicating that the current ImpRec setting is close to the optimum. Values in italic font are rounded off to the nearest integer value.

Step	$ALPHA$	$PENALTY$	$START$	$LEVEL$	Resp.
0	0.99	0.80	76	3	0.50625
1	0.99	0.80	*75.60777*	*3.980581*	0.5
2	0.99	0.80	*75.21554*	*4.961161*	0.5
3	0.99	0.80	*74.8233*	*5.941742*	0.48125
4	0.99	0.80	*74.43107*	*6.922323*	0.45625

```
116 Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
117
118 Multiple R-squared:  0.9905,     Adjusted R-squared:  0.9833
119 F-statistic: 138.7 on 3 and 4 DF,  p-value: 0.0001695
120
121 Analysis of Variance Table
122
123 Response: resp
124                Df     Sum Sq     Mean Sq F value     Pr(>F)
125 FO(x2, x3, x4)  3 0.00203125 0.00067708  138.67 0.0001695
126 Residuals       4 0.00001953 0.00000488
127 Lack of fit     4 0.00001953 0.00000488
128 Pure error      0 0.00000000
129
130 Direction of steepest ascent (at radius 1):
131            x2            x3            x4
132  4.233906e-16 -1.961161e-01  9.805807e-01
133
134 Corresponding increment in original units:
135    penalty       start       level
136  0.0000000 -0.3922323  0.9805807
```

C: CCD and a Second-order Polynomial Model

The final step in RSM is to fit a second-order polynomial model to the region close to the maximum, and to locate the stationary point. The most popular design for fitting a second-order model is CCD [342, pp. 501] (cf. C in Fig. 1). In traditional DoE, it is recommended to conduct three to five experimental runs at the center point. When tuning an SE tool, we do not need more than one, thus the only choice for the experimental design is the distance of the *axial runs*. As presented in Figure 1, we recommend a rotatable design, i.e., that all settings in the tuning experiment should be at the same distance from the center point.

In the CCD experiment for tuning ImpRec, we focus on the two parameters $START$ and $LEVEL$. Listing V.5:134 shows that these two parameters dwarf $PENALTY$ in this region. Furthermore, the parameter $ALPHA$ is already at

Table 7: Central composite design for the two parameters $START$ and $LEVEL$. Values in italic font are rounded off to the nearest integer value.

	Coded variables		Natural variables		
Exp. Run	$x3$	$x4$	$START$	$LEVEL$	Resp.
1	-1	-1	72	2	0.453125
2	1	-1	80	2	0.4625
3	-1	1	72	4	0.490625
4	1	1	80	4	0.490625
5	0	0	76	3	0.50625
6	-1.414	0	*70.344*	3	0.490625
7	+1.414	0	*81.656*	3	0.48125
8	0	-1.414	76	*4.414*	0.5
9	0	+1.414	76	*1.586*	0.465625

its maximum value. Table 7 shows the CCD experiment and the corresponding responses. We store the table, except the coded variables, in *ccd.csv*. Listing V.6 shows the analysis of the results, including the coding transformation of the parameters. In the final step in Phase 3 of TuneR, the outcome of the CCD experiment is analyzed.

Listing V.6: Using *rsm* [294] to fit a second-order model of the response surface in the vicinity of the optimal response.

```
> ccd <- read.csv("ccd.csv")
> ccd_coded <- coded.data(ccd, x3~(start-76)/2, x4~(level-3)/1)
> ccd_model <- rsm(resp~SO(x3, x4), data=ccd_coded)
> summary(ccd_model)

Call:
rsm(formula = resp ~ SO(x3, x4), data = ccd_coded)

              Estimate   Std. Error  t value  Pr(>|t|)
(Intercept)  0.50614821  0.00268418 188.5672 4.745e-09 ***
x3          -0.00027574  0.00068784  -0.4009 0.7090065
x4           0.01666667  0.00163740  10.1788 0.0005247 ***
x3:x4       -0.00117188  0.00100270  -1.1687 0.3074152
x3^2        -0.00223432  0.00039648  -5.6354 0.0048794 **
x4^2        -0.02310668  0.00268905  -8.5929 0.0010078 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Analysis of Variance Table

Response: resp
              Df     Sum Sq     Mean Sq    F value    Pr(>F)
FO(x3, x4)     2 0.00166925 0.00083463 5.1884e+01  0.001378
TWI(x3, x4)    1 0.00002197 0.00002197 1.3659e+00  0.307415
PQ(x3, x4)     2 0.00137822 0.00068911 4.2838e+01  0.001990
```

```
Residuals      4 0.00006435 0.00001609
Lack of fit    3 0.00006435 0.00002145 4.4547e+29 1.101e-15
Pure error     1 0.00000000 0.00000000

Stationary point of response surface:
         x3          x4
-0.1573278  0.3646356

Stationary point in original units:
     start      level
75.685344  3.364636

Eigenanalysis:
$values
[1] -0.002217888 -0.023123113

$vectors
           [,1]       [,2]
x3 -0.9996068 0.0280393
x4  0.0280393 0.9996068
```

D: Evaluate Stationary Point

The purpose of the previous TuneR step was to locate a stationary point in the vicinity of the optimal setting. The stationary point can either represent a maximum response, a minimum response, or a saddle point. The nature of the stationary point is given by the signs of the *eigenvalues*: for a maximum response all are negative, and for a minimum response all are positive. Thus, if the eigenanalysis resulted in a maximum point, the tuning experiments have resulted in a pin-pointed optimal setting for the SE tool. If the eigenvalues have different signs on the other hand, then the CCD experiment located a saddle point. The experimenter should then perform *ridge analysis* [357], i.e., conducting additional experimental runs following the ridge in both directions.

Regarding the stationary point identified for the tuning of ImpRec, it is located close to $START = 76, LEVEL = 3$ as shown in Listing V.6:170. The eigenanalysis gives that it represents a maximum point (cf. Listing V.6:174). Figure 8 shows a visualization[6] using *visreg* [84] of the response surface in this region. visualizes the response surface in this region, confirming that a setting representing a maximum response has been identified. Thus, we conclude the parameter tuning of ImpRec as follows: $ALPHA = 0.99, PENALTY = 0.80, START = 76, LEVEL = 3$. Using the new parameter setting of ImpRec, we obtain $Rc@20 = 0.50625$ compared to $Rc@20 = 0.41875$ using the default settings of ImpRec ($ALPHA = 0.83, PENALTY = 0.2, START = 17, LEVEL = 7$). Applying TuneR has thus improved the Rc@20 for ImpRec by 20.9% on this particular dataset.

[6] R command: $> visreg2d(ccd\_model, "x3", "x4", plot.type = "persp")$

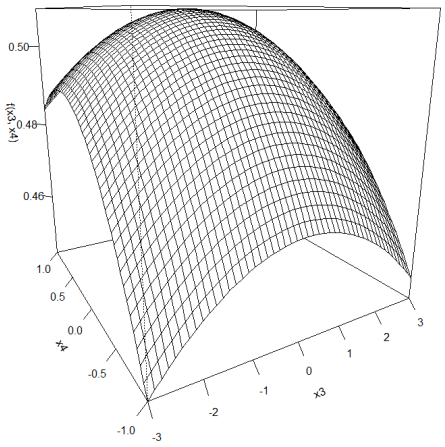


Figure 8: Visualization of the response surface wrt. $x3$ and $x4$, i.e., $START$ and $LEVEL$ in coded variables. $ALPHA$ and $PENALTY$ are fixed to 0.99 and 0.80, respectively. f(x3, x4) shows the response.

5.4 Evaluate the Setting

The final activity in TuneR is to perform an evaluation of the new parameter setting. Optimization based on a single response metric might result in a far too naïve perspective, thus a more holistic analysis must be employed to determine the value of the new parameter setting. Numbers typically do not cover all aspects of a studied phenomenon [324], and there is a risk that the experimenter pushes the setting too much based on quantitative metrics, squeezing percentages without considering overall values of the process the SE tool is intended to support. The final activity of TuneR aims at taking a step back, and considering the bigger picture.

Figure 9 shows a comparison of the ImpRec evaluation using the default setting (dashed line) and the tuned setting (solid line). The four subplots show the cut-off, N, of the ranked output list on the x-axis, and the following IR measures:

A: **Precision**, displaying the decrease that is characteristic to IR evaluations [89].

B: **Recall**, including the target metric for the tuning experiments: Rc@20.

C: **F_1-score**, the (balanced) harmonic mean of recall and precision.

D: **MAP**, an IR measure that combines recall with the performance of ranking.

The evaluation reveals that while the tuning has resulted in increases with regard to recall (cf. Fig. 9, subplot B), the improvements have been paid by other metrics. Indeed, TuneR has increased the target metric Rc@20 by 20.9%. Moreover, the response for higher N is even higher, reaching as high as 0.544 for

Rc@43-50 (an increase by 27.0%). However, at low N the default setting actually reaches a higher recall, and first at Rc@11 the tuned setting becomes becomes better. To further add to the discussion, the three subplots A, C, and D all show that the default setting outperforms the tuned setting. For MAP@N, the difference between the default setting and the tuned setting actually increases for large N.

The evaluation of the tuned parameter setting for ImpRec, and the identified trade-off, show the importance of the final step in TuneR. It is not at all clear from Figure 9 whether the new parameter setting would benefit an engineer working with ImpRec. While we have carefully selected the response metrics, the trade-off appears to be bigger than expected. Not only is the trade-off between recall and precision evident, but also the trade-off within Rc@N; only after the cut-off N = 11 the recall benefits from the new setting. Our work is an example of a purely quantitative *in silico* evaluation, conducted as computer experiments without considering the full operational context of ImpRec [77]. To fully understand how switching between the two settings affect the *utility* [32] of ImpRec, human engineers actually working with the tool must be studied. We report from such an *in situ* study in another paper [80], in which we deployed ImpRec in two development teams in industry, one with the default setting and one with the tuned setting.

6 Tuning ImpRec Using Exhaustive Search

If the screening experiments of TuneR (Phase 2) fails to identify actionable regularities in the response surface, i.e., there is considerable lack of fit for both first and second-order models, the experimenter might decide to design an experiment of a more exhaustive nature. However, as an exhaustive amount of experimental runs is likely to be computationally expensive, a first try should be to investigate if a low-order polynomial model fit for the promising part of the response surface. If that is the case, Phase 3 could still be applicable in that specific region of the response surface. Otherwise, at least if the set of critical parameters has been reduced, a more exhaustive space-filling design (i.e., a brute force approach [358]) might be the remaining option to find a tuned setting. The purpose of this section is twofold. First, we present the design of a fine-granular space-filling design for tuning ImpRec. Second, the result of the exhaustive search acts as a proof-of-concept of TuneR, as we compare the results to the outcome from Phase 3.

For tuning of ImpRec, we design a uniform space-filling design. Table 8 shows the levels we explore in the experimental runs. The screening experiment described in Section 5.2 shows that $ALPHA$ appears to be more important than $PENALTY$, thus we study it with a finer granularity. $START$ and $LEVEL$ are both positive integer parameters, and we choose to explore them starting from their lowest possible values. As the nature of the issue-issue links is unlikely to result in issue chains longer than five, setting the highest value to 9 is already a conservative

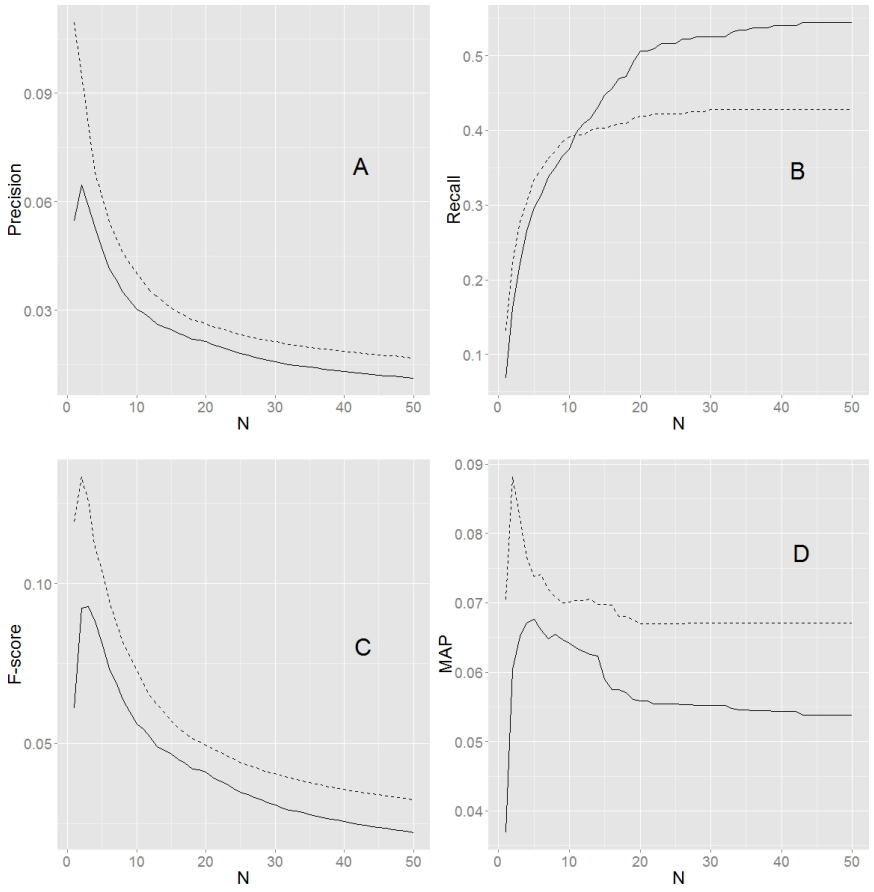


Figure 9: Comparison of ImpRec output with default settings (dashed line) and tuned settings (solid line). Subplots clockwise from the first quadrant: Recall@N, MAP@N, F_1-score@N, and Precision@N,

choice. The potential of large $START$ on the other hand is less clear, but Figure 6 suggests that values between 16 and 128 result in the best Rc@20. However, large $START$ require infeasible execution times, thus we restrict the parameter to 90 for practical reasons.

Table 9 shows the best results from running the exhaustive tuning experiments. In total, the experiments required 1,253 hours (about 52 days) to complete on a desktop computer[7], with an average of 24 s per experimental run. The best result we obtain in the exhaustive experiments is Rc@20=0.5375, a response we get from

[7] Intel Core i5-2500K quad-core CPU 3.30 GHz with 8 GB RAM.

Table 8: Uniform space-filling design for exhaustive approach to tuning of ImpRec. The design requires 187,110 experimental runs, compared to 3,430 in the screening experiment (cf. Table 2).

Parameter	#Levels	Values
$ALPHA$	21	0.01, 0.05, 0.10, 0.15, ..., 0.95, 0.99
$PENALTY$	11	0.01, 0.5, 1, 1.5, ..., 5
$START$	90	1, 2, 3, ..., 90
$LEVEL$	9	1, 2, ..., 9

Table 9: Top 10 results from the exhaustive experiment. The third column shows how many different settings that yield the response.

	Rc@20	#Settings
1	0.5375	12
2	0.534375	72
3	0.53125	60
4	0.528125	72
5	0.525	108
6	0.521875	238
7	0.51875	96
8	0.515625	120
9	0.5125	238
10	0.509375	83

12 different settings, a value that is 6.2% better than what we found using the three main phases of TuneR (Rc@20=0.50625). By looking at the 12 settings yielding Rc@20=0.5375, we note that $START = 51$ and $LEVEL = 3$ provide the best results. However, regarding the two remaining parameters, the pattern is less clear; $ALPHA$ varies from 0.6 to 0.99, and $PENALTY$ is either at low range (0.5 or 1.5) or at high range (4.5 or 5). Figure 10 summarizes the exhaustive experiment by presenting the distribution of responses per setting, as well as the execution times.

7 Discussion

Finding feasible parameter settings for SE tools is a challenging, but important, activity. SE tools are often highly configurable through parameters, but there is typically no silver bullet; there is not one default parameter setting that is optimal for all contexts. However, often advanced approaches are implemented in state-of-the-art SE tools. As a result of the tools' inherent complexity, academic researchers have published numerous papers on how to improve tool output by

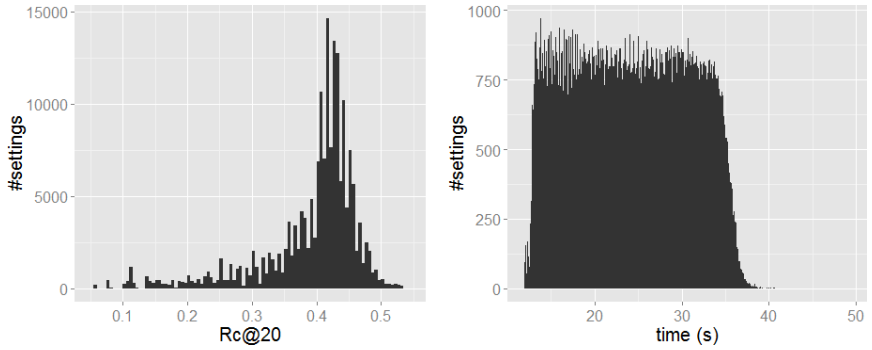


Figure 10: Distribution of results from the exhaustive experiment. The y-axes show the number of settings that resulted in the output. The left figure displays Rc@20, and the right figure shows the execution time.

trying different settings and tuning internal algorithms. Consequently, SE tool developers cannot expect end users to understand all the intricate details of their implementations. Instead we argue that applied researchers need to provide guidelines to stimulate dissemination of SE tools in industry, i.e., to support transfer of research prototypes from academia to industry.

An established approach to improve processes is to use experiments. However, traditional DoE was developed for physical processes, much different from the application of SE tools. In this paper we introduced TuneR, a framework for tuning SE tools, using a combination of space-filling designs and RSM. Several researchers have presented advice and guidelines on how to tune various SE tools, but they typically address a specific family of tools, e.g., SBSE [25, 188], evolutionary software testing [130], LDA for feature location [58], and trace retrieval [313]. TuneR is instead a general framework, that can be applied for most types of SE tools.

As a proof-of-concept, and as a demonstration of TuneR's ease of use, we presented a detailed step-by-step tuning of the RSSE ImpRec. Using TuneR we obtain a considerable increase in the response variable of ImpRec, even though we considered the default setting already good enough for tool deployment in industry (see our industrial case study for further details [80]). We selected the default setting[8] based on *ad hoc* tuning during development of ImpRec, but using TuneR resulted in a 20.9% higher response, i.e., an improvement from Rc@20=0.41875 to Rc@20=0.50625. Thus, in contrast to the Arcuri and Fraser's inconclusive results from tuning an SBSE tool [25], we demonstrate that RSM can be a component in successful tuning of SE tools.

[8]Default setting: $ALPHA = 0.83, PENALTY = 0.2, START = 17, LEVEL = 7$

Applying TuneR to tune an SE tool provides insights beyond what a feasible parameter setting. Thanks to the screening phase, TuneR identifies the most important parameters, both in terms of main effects and interaction effects. Especially interaction effects is missed when tuning tools using less structured experimental designs, e.g., COST analysis and *ad hoc* tuning [165, pp. 211] [342, pp. 4]. During tuning of ImpRec, we found that two interactions were significant: 1) positive interaction between $ALPHA$ and $START$, and 2) negative interaction between $START$ and $LEVEL$. Thus, if a high number of issue reports are used as starting points, then the ranking function should give more weight to the centrality measure than the textual similarity. Furthermore, if the number of starting points is high, then the number of links to follow in the knowledge base should be decreased.

Although resulting in a considerable improvement in the response, we found that the tuned setting[9] obtained from TuneR still was not optimal. Using exhaustive experiments, we identified settings that yield even higher responses, reaching as high as Rc@20=0.5375. However, running exhaustive experiments come at a high computational cost, and it is not certain that there is enough return on investment. In our example, we used more than 50 days of computation time for the exhaustive experiments, in total conducting 187,110 experimental runs, to find a 6.2% higher response (Rc@0=0.5375) compared to the TuneR setting. Furthermore, we explored only four parameters in the exhaustive experiments. For other SE tools the number of parameters might be higher, and the combinatorial explosion would quickly lead to infeasible exhaustive experimental designs. To mitigate this problem, the screening phase of TuneR could be used to identify the dominating parameters, in line with common practice in traditional DoE [165, 342].

The exhaustive experiments revealed 12 different settings yielding the top response. A clear pattern in the 12 settings was found; to obtain the best results, $START$ and $LEVEL$ were set to 51 and 3, respectively. At the same time however, $ALPHA$ and $PENALTY$ could be set to several different combinations of values. Based on the screening phase of TuneR, we concluded that $ALPHA$ should be set to high values, as "centrality values are more important than textual similarity, i.e., previously impacted artifacts are likely to be impacted again" (see Section 5.2). In hindsight, with the knowledge obtained from the exhaustive experiment, it appears that early fixing $ALPHA$ to 0.99 was not necessarily the right decision, as high responses apparently can be obtained for a range of $ALPHA$ values. Experimentation is an iterative process, and the experimenter's knowledge gradually increases. Based on the updated understanding of $ALPHA$, a next step could be to do another TuneR screening focusing on $0.6 \leq ALPHA \leq 0.99$.

We acknowledge two main threats to the validity of the tuned ImpRec setting we obtain through TuneR. First, there is always a threat that *focusing on a single response metric might be an oversimplification*, as discussed in Section 5.1. In Section 5.4, we show that while the tuned setting leads to an improved response in Rc@20, with regard to most other metrics we study in the evaluation of the

[9] Tuned setting: $ALPHA = 0.99, PENALTY = 0.80, START = 76, LEVEL = 3$

new setting, the output was better for the default setting. Whether Rc@20 is the best target metric is not certain, even though we posit that it reflects an important quality aspect of ImpRec, resulting in maximization of true impact among a manageable amount of recommendations. An alternative response metric could be MAP@20, also reported in the evaluation in Section 5.4, a metric that also considers the ranking of the true output among the top-20 recommendations. We stress that it is important to validate the response metric from the start, otherwise TuneR will move the setting in a direction that does not bring value.

Second, while we carefully selected four parameters for the tuning experiments, *there might be additional important parameters at play*. For example, the IR approach we apply in ImpRec could be adjusted in numerous ways, yet we consider the involved variation points as fixed. Apache Lucene, the integrated IR solution, is highly configurable, but as we have successfully used it for a similar task before (duplicate detection of issue reports [78]), we made the assumption that it performs well out-of-the-box. Other potentially useful approaches related to IR, which we did not explore in this paper, is to perform further preprocessing, e.g., stop word removal, stemming, and dimensionality reduction. However, as TuneR resulted in an increased response, also close to what the exhaustive experiment yielded, we argue that our selection of parameters was valid.

Furthermore, there are also some threats to the validity of the overall TuneR framework. While our goal when developing TuneR was to present a framework generally applicable to tuning of SE tools, the *external validity* [477] *of the approach is still uncertain*. We have only presented one single proof-of-concept, i.e., the tuning of the RSSE ImpRec, thus we need to conduct additional tuning experiments, with other SE tools, to verify the generalizability. We plan to continue evolving TuneR, and two involved activities we particularly want to focus on improving are: 1) guidelines regarding *parameter subset selection* when fitting low-order polynomial models during screening (Section 5.2), and 2) the *step size selection* in the RSM phase (Section 5.3). Finally, we argue that TuneR is easy to use, especially since we present hands-on examples in R, but the only way to validate the usability is by letting others try the framework.

8 Conclusion

In this paper we have presented TuneR, an experiment framework for tuning Software Engineering (SE) tools. TuneR build on methods from Design of Experiments (DoE) and Design of Computer Experiments (DoCE), two established fields with numerous successful applications in various engineering disciplines [236]. However, both DoE and DoCE have been developed to address experiments on phenomena with a representation in the physical world, either directly (DoE) or indirectly through computer models (DoCE). We have discussed how tuning of

SE tools is different from traditional experimentation, and how *TuneR combines space-filling designs and factorial designs to identify a feasible parameter setting*.

As a proof-of-concept, we applied TuneR to tune ImpRec, a recommendation system for change impact analysis, to a specific proprietary context. For all TuneR steps, we have provided detailed instructions on how to analyze the experimental output using various R packages. Using TuneR, we *increased the accuracy of the ImpRec recommendations by 20.9%* with regard to recall among the top-20 candidates. To validate the tuned setting, we also applied a more exhaustive space-filling design, trying in total 187,110 parameter settings. We found parameter settings yielding a 6% higher response, but running the experiment required more than 50 days of computation time. Thus, we consider the proof-of-concept successful, as TuneR resulted in a similar response in a fraction of the time.

A major threat when tuning an SE tool is that the selected response metric, i.e., the target for optimization, does not fully capture the overall value of the tool. Optimizing a response might come at a price; increases in one metric might be paid by decreases in other metrics. The tuning of ImpRec is an example of this trade-off, and we show how precision, F_1-score, and mean average precision decrease with the new tuned setting. Even recall at lower cut-off points, i.e., when considering ten or fewer recommendations from ImpRec, yields decreased results with the tuned parameter setting. From this observation, we *stress the importance of carefully selecting the response metric*, and to *properly evaluate the consequences of the tuned parameter setting*, before deploying the tuned SE tool.

Acknowledgements

This work was funded by the Industrial Excellence Center EASE - Embedded Applications Software Engineering[10].

[10] http://ease.cs.lth.se

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] A. Abadi, M. Nisenson, and Y. Simionovici. A Traceability Technique for Specifications. In *Proc. of the International Conference on Program Comprehension*, pages 103–112, 2008.

[2] W. AbdelMoez, M. Kholief, and F. Elsalmy. Improving Bug Fix-Time Prediction Model by Filtering Out Outliers. In *Proc. of the 1st International Conference on Technological Advances in Electrical, Electronics and Computer Engineering*, pages 359–364, 2013.

[3] M. Aberdour. Achieving Quality in Open-Source Software. *IEEE Software*, 24(1):58–64, 2007.

[4] R. Abreu, P. Zoeteweij, and A. van Gemund. An Evaluation of Similarity Coefficients for Software Fault Localization. In *Proc. of the 12th Pacific Rim International Symposium on Dependable Computing*, pages 39–46, 2006.

[5] M. Acharya and B. Robinson. Practical Change Impact Analysis Based on Static Program Slicing for Industrial Software Systems. In *Proc. of the 33rd International Conference on Software Engineering*, pages 746–755, 2011.

[6] S. Ahsan, J. Ferzund, and F. Wotawa. Automatic Software Bug Triage System (BTS) Based on Latent Semantic Indexing and Support Vector Machine. In *Proc. of the 4h International Conference on Software Engineering Advances*, pages 216–221, 2009.

[7] J. Aitchison, A. Gilchrist, and D. Bawden. *Thesaurus Construction and Use: A Practical Manual*. Routledge, 4th edition, 2000.

[8] M. Alenezi, K. Magel, and S. Banitaan. Efficient Bug Triaging Using Text Mining. *Journal of Software*, 8(9):2185–2190, 2013.

[9] N. Ali, Y. Guéhéneuc, and G. Antoniol. Trust-Based Requirements Traceability. In *Proc. of the 19th International Conference on Program Comprehension*, pages 111–120, 2011.

[10] N. Ali, Y. Guéhéneuc, and G. Antoniol. Factors Impacting the Inputs of Traceability Recovery Approaches. In J. Cleland-Huang, O. Gotel, and A. Zisman, editors, *Software and Systems Traceability*, pages 99–127. Springer, 2012.

[11] R. Alshammari and A. Zincir-Heywood. Machine Learning Based Encrypted Traffic Classification: Identifying SSH and Skype. In *Proc. of the 2nd Symposium on Computational Intelligence for Security and Defense Applications*, pages 1–8, 2009.

[12] M. Alvesson and K. Sköldberg. *Reflexive Methodology: New Vistas for Qualitative Research*. Sage Publications, 2000.

[13] A. Amamra, C. Talhi, J. Robert, and M. Hamiche. Enhancing Smartphone Malware Detection Performance by Applying Machine Learning Hybrid Classifiers. In *Proc. of the International Conference on Advanced Software Engineering & Its Applications and Proc. of the International Conference on Disaster Recovery and Business Continuity*, pages 131–137, 2012.

[14] C. Andersen and R. Bro. Variable Selection in Regression - A Tutorial. *Journal of Chemometrics*, 24(11-12):728–737, 2010.

[15] N. Ansari and E. Hou. *Computational Intelligence for Optimization*. Springer, 2012.

[16] G. Antoniol, G. Canfora, G. Casazza, and A. De Lucia. Information Retrieval Models for Recovering Traceability Links between Code and Documentation. In *Proc. of the 16th International Conference on Software Maintenance*, pages 40–49, 2000.

[17] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo. Tracing Object-Oriented Code Into Functional Requirements. In *Proc. of the 8th International Workshop on Program Comprehension*, pages 79–86, 2000.

[18] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo. Recovering Traceability Links between Code and Documentation. *Transactions on Software Engineering*, 28(4):970–983, 2002.

[19] G. Antoniol, G. Canfora, A. De Lucia, and E. Merlo. Recovering Code to Documentation Links in OO Systems. In *Proc. of the 6th Working Conference on Reverse Engineering*, pages 136–144, 1999.

[20] G. Antoniol, A. Potrich, P. Tonella, and R. Fiutem. Evolving Object Oriented Design to Improve Code Traceability. In *Proc. of the 7th International Workshop on Program Comprehension*, pages 151–160, 1999.

[21] J. Anvik. *Assisting Bug Report Triage through Recommendation*. PhD Thesis, University of British Columbia, 2007.

[22] J. Anvik, L. Hiew, and G. Murphy. Coping with an Open Bug Repository. In *Proc. of the 2005 OOPSLA Workshop on Eclipse Technology eXchange*, pages 35–39, 2005.

[23] J. Anvik, L. Hiew, and G. Murphy. Who Should Fix this Bug? In *Proc. of the 28th International Conference on Software Engineering*, pages 361–370, 2006.

[24] J. Anvik and G. Murphy. Reducing the Effort of Bug Report Triage: Recommenders for Development-Oriented Decisions. *Transactions on Software Engineering and Methodology*, 20(3):1–35, 2011.

[25] A. Arcuri and G. Fraser. Parameter Tuning or Default Values? An Empirical Investigation in Search-based Software Engineering. *Empirical Software Engineering*, 18(3):594–623, 2013.

[26] S. Arlot and A. Celisse. A Survey of Cross-Validation Procedures for Model Selection. *Statistics Surveys*, 4:40–79, 2010.

[27] R. Arnold and S. Bohner. Impact Analysis - Towards a Framework for Comparison. In *Proc. of the 9th Conference on Software Maintenance*, pages 292–301, 1993.

[28] U. Asklund and L. Bendix. A Study of Configuration Management in Open Source Software Projects. *IEE Proceedings - Software*, 149(1):40–46, 2002.

[29] N. Assawamekin, T. Sunetnanta, and C. Pluempitiwiriyawej. Ontology-Based Multiperspective Requirements Traceability Framework. *Knowledge and Information Systems*, 25(3):493–522, 2010.

[30] H. Asuncion, F. François, and R. Taylor. An End-to-End Industrial Software Traceability Tool. In *Proc. of the 6th Joint Meeting of the European Software Engineering Conference and the Symposium on The Foundations of Software Engineering*, pages 115–124, 2007.

[31] H. Asuncion and R. Taylor. Automated Techniques for Capturing Custom Traceability Links Across Heterogeneous Artifacts. In J. Cleland-Huang, O. Gotel, and A. Zisman, editors, *Software and Systems Traceability*, pages 129–146. Springer, 2012.

[32] I. Avazpour, T. Pitakrat, L. Grunske, and J. Grundy. Dimensions and Metrics for Evaluating Recommendation Systems. In M. Robillard, W. Maalej, R. Walker, and T. Zimmermann, editors, *Recommendation Systems in Software Engineering*, pages 245–273. Springer, 2014.

[33] K. Ayari, P. Meshkinfam, G. Antoniol, and M. Di Penta. Threats on Building Models from CVS and Bugzilla Repositories: The Mozilla Case Study.

In *Proc. of the 17th Conference of the Center for Advanced Studies on Collaborative Research*, pages 215–228, 2007.

[34] N. Ayewah, D. Hovemeyer, J. Morgenthaler, J. Penix, and W. Pugh. Using Static Analysis to Find Bugs. *IEEE Software*, 25(5):22–29, 2008.

[35] A. Bacchelli, M. Lanza, and R. Robbes. Linking E-mails and Source Code Artifacts. In *Proc. of the 32nd International Conference on Software Engineering*, pages 375–384, 2010.

[36] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval: The Concepts and Technology behind Search*. Addison-Wesley, 2nd edition, 2011.

[37] M. Banko and E. Brill. Scaling to Very Very Large Corpora for Natural Language Disambiguation. In *Proc. of the 39th Annual Meeting on Association for Computational Linguistics*, pages 26–33, 2001.

[38] A. Barbolla and J. Corredera. Critical Factors for Success in University-Industry Research Projects. *Technology Analysis & Strategic Management*, 21(5):599–616, 2009.

[39] Z. Barmi, A. Ebrahimi, and R. Feldt. Alignment of Requirements Specification and Testing: A Systematic Mapping Study. In *Proc. of the ICST Workshop on Requirements and Validation, Verification and Testing*, pages 476–485, 2011.

[40] T. Bartz-Beielstein, C. Lasarczyk, and M. Preuss. The Sequential Parameter Optimization Toolbox. In T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss, editors, *Experimental Methods for the Analysis of Optimization Algorithms*, pages 337–362. Springer, 2010.

[41] V. Basili. Software Modeling and Measurement: The Goal/Question/Metric Paradigm. Technical Report CS-TR-2956, University of Maryland, 1992.

[42] V. Basili. The Role of Controlled Experiments in Software Engineering Research. In V. Basili, D. Rombach, K. Schneider, B. Kitchenham, D. Pfahl, and R. Selby, editors, *Empirical Software Engineering Issues. Critical Assessment and Future Directions*, pages 33–37. Springer, 2007.

[43] V. Basili, R. Selby, and D. Hutchens. Experimentation in Software Engineering. *Transactions on Software Engineering*, 12(7):733–743, 1986.

[44] O. Baysal, M. Godfrey, and R. Cohen. A Bug You Like: A Framework for Automated Assignment of Bugs. In *Proc. of the 17th International Conference on Program Comprehension*, pages 297–298, 2009.

[45] M. Baz, B. Hunsaker, P. Brooks, and A. Gosavi. Automated Tuning of Optimization Software Parameters. Technical Report, Dept. of Industrial Engineering, University of Pittsburgh, 2007.

[46] K. Beck. *Test-Driven Development: By Example*. Addison-Wesley, 2003.

[47] E. Ben Charrada, D. Caspar, C. Jeanneret, and M. Glinz. Towards A Benchmark for Traceability. In *Proc. of the 12th International Workshop on Principles on Software Evolution*, pages 21–30, 2011.

[48] J. Berlik, S. Dharmadhikari, M. Harding, and N. Singh. System and Method for Maintaining Requirements Traceability, US Patent 8191044 B1, 2012.

[49] R. Berntsson Svensson, Y. Sprockel, B. Regnell, and S. Brinkkemper. Setting Quality Targets for Coming Releases with QUPER: An Industrial Case Study. *Requirements Engineering*, 17(4):283–298, 2012.

[50] D. Berry. *The Philosophy of Software: Code and Mediation in the Digital Age*. Palgrave Macmillan, 2011.

[51] D. Bertram, A. Voida, S. Greenberg, and R. Walker. Communication, Collaboration, and Bugs: The Social Nature of Issue Tracking in Small, Collocated Teams. In *Proc. of the 13th Conference on Computer Supported Cooperative Work*, pages 291–300, 2010.

[52] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann. What Makes a Good Bug Report? In *Proc. of the 16th International Symposium on Foundations of Software Engineering*, pages 308–318, 2008.

[53] N. Bettenburg, R. Premraj, T. Zimmermann, and K. Sunghun. Duplicate Bug Reports Considered Harmful... Really? In *Proc. of the 24th International Conference on Software Maintenance*, pages 337–345, 2008.

[54] J. Bezanson, S. Karpinski, V. Shah, and A. Edelman. Julia: A Fast Dynamic Language for Technical Computing. 2012. arXiv: 1209.5145.

[55] P. Bhattacharya, I. Neamtiu, and C. Shelton. Automated, Highly-accurate, Bug Assignment Using Machine Learning and Tossing Graphs. *Journal of Systems and Software*, 85(10):2275–2292, 2012.

[56] A. Bianchi, A. Fasolino, and G. Visaggio. An Exploratory Case Study of the Maintenance Effectiveness of Traceability Models. In *Proc. of the 8th International Workshop on Program Comprehension*, pages 149–158, 2000.

[57] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. MOA: Massive Online Analysis. *Journal of Machine Learning Research*, 11:1601–1604, 2010.

[58] L. Biggers, C. Bocovich, R. Capshaw, B. Eddy, L. Etzkorn, and N. Kraft. Configuring Latent Dirichlet Allocation Based Feature Location. *Empirical Software Engineering*, 19(3):465–500, 2014.

[59] D. Binkley and D. Lawrie. Information Retrieval Applications in Software Maintenance and Evolution. In J. Marciniak, editor, *Encyclopedia of Software Engineering*. Taylor & Francis, 2nd edition, 2010.

[60] D. Binkley and D. Lawrie. Learning to Rank Improves IR in SE. In *Proc. of the 30th International Conference on Software Maintenance and Evolution*, pages 441–445, 2014.

[61] M. Birattari. *Tuning Metaheuristics - A Machine Learning Perspective*. Springer, 2009.

[62] C. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[63] E. Bjarnason, K. Smolander, E. Engström, and P. Runeson. Alignment Practices Affect Distances in Software Development: A Theory and a Model. In *Proc. of the 3rd SEMAT Workshop on General Theories of Software Engineering*, pages 21–31, 2014.

[64] D. Blei and J. Lafferty. A Correlated Topic Model of Science. *Annals of Applied Statistics*, 1(1):17–35, 2007.

[65] D. Blei, A. Ng, and M. Jordan. Latent Dirichlet Allocation. *The Journal of Machine Learning Research*, 3(4-5):993–1022, 2003.

[66] B. Boehm. Software Engineering. *Transactions on Computers*, 25(12):1226–1241, 1976.

[67] S. Bohner. *Software Change Impact Analysis*. IEEE Computer Society Press, 1996.

[68] S. Bohner. Software Change Impacts - An Evolving Perspective. In *Proc. of the 18th International Conference on Software Maintenance*, pages 263–272, 2002.

[69] M. Borg. In Vivo Evaluation of Large-Scale IR-Based Traceability Recovery. In *Proc. of the 15th European Conference on Software Maintenance and Reengineering*, pages 365–368, 2011.

[70] M. Borg. TuneR: A Framework for Tuning Software Engineering Tools with Hands-On Instructions in R. *Submitted to a journal*, 2015.

[71] M. Borg, O. Gotel, and K. Wnuk. Enabling Traceability Reuse for Impact Analyses: A Feasibility Study in a Safety Context. In *Proc. of the 7th International Workshop on Traceability in Emerging Forms of Software Engineering*, 2013.

[72] M. Borg and D. Pfahl. Do Better IR Tools Improve the Accuracy of Engineers' Traceability Recovery? In *Proc. of the International Workshop on Machine Learning Technologies in Software Engineering*, pages 27–34, 2011.

[73] M. Borg, D. Pfahl, and P. Runeson. Analyzing Networks of Issue Reports. In *Proc. of the 17th European Conference on Software Maintenance and Reengineering*, pages 79–88, 2013.

[74] M. Borg and P. Runeson. IR in Software Traceability: From a Bird's Eye View. In *Proc of the 7th International Symposium on Empirical Software Engineering and Measurement*, pages 243–246, 2013.

[75] M. Borg and P. Runeson. Changes, Evolution and Bugs - Recommendation Systems for Issue Management. In M. Robillard, W. Maalej, R. Walker, and T. Zimmermann, editors, *Recommendation Systems in Software Engineering*, pages 477–509. Springer, 2014.

[76] M. Borg, P. Runeson, and A. Ardö. Recovering from a Decade: A Systematic Mapping of Information Retrieval Approaches to Software Traceability. *Empirical Software Engineering*, 19(6):1565–1616, 2014.

[77] M. Borg, P. Runeson, and L. Brodén. Evaluation of Traceability Recovery in Context: A Taxonomy for Information Retrieval Tools. In *Proc. of the 16th International Conference on Evaluation & Assessment in Software Engineering*, pages 111–120, 2012.

[78] M. Borg, P. Runeson, J. Johansson, and M. Mäntylä. A Replicated Study on Duplicate Detection: Using Apache Lucene to Search Among Android Defects. In *Proc. of the 8th International Symposium on Empirical Software Engineering and Measurement*, 2014.

[79] M. Borg, K. Wnuk, and D. Pfahl. Industrial Comparability of Student Artifacts in Traceability Recovery Research - An Exploratory Survey. In *Proc. of the 16th European Conference on Software Maintenance and Reengineering*, pages 181–190, 2012.

[80] 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. *Submitted to a journal*, 2015.

[81] M. Borillo, A. Borillo, N. Castell, D. Latour, Y. Toussaint, and M. Felisa Verdejo. Applying Linguistic Engineering to Spatial Software Engineering: The Traceability Problem. In *Proc. of the 10th European Conference on Artificial Intelligence*, pages 593–595, 1992.

[82] G. Box, S. Hunter, and W. Hunter. *Statistics for Experimenters: Design, Innovation, and Discovery*. Wiley, 2nd edition, 2005.

[83] M. Bras and Y. Toussaint. Artificial Intelligence Tools for Software Engineering: Processing Natural Language Requirements. In *Applications of Artificial Intelligence in Engineering*, pages 275–290, 1993.

[84] P. Breheny and W. Burchett. Visualization of Regression Models Using visreg. Technical Report, University of Kentucky, 2013.

[85] L. Breiman. Bagging Predictors. *Machine Learning*, 24(2):123–140, 1996.

[86] P. Brereton, B. Kitchenham, D. Budgen, M. Turner, and M. Khalil. Lessons from Applying the Systematic Literature Review Process within the Software Engineering Domain. *Journal of Systems and Software*, 80(4):571–583, 2007.

[87] L. Briand, Y. Labiche, L. O'Sullivan, and M. Sówka. Automated Impact Analysis of UML Models. *Journal of Systems and Software*, 79(3):339–352, 2006.

[88] M. Brown and D. Goldenson. Measurement and Analysis: What Can and Does Go Wrong? In *Proc. of the 10th International Symposium on Software Metrics*, pages 131–138, 2004.

[89] M. Buckland and F. Gey. The Relationship between Recall and Precision. *Journal of the American Society for Information Science*, 45(1):12–19, 1994.

[90] P. Burman, E. Chow, and D. Nolan. A Cross-Validatory Method for Dependent Data. *Biometrika*, 81(2):351–358, 1994.

[91] G. Canfora and L. Cerulo. Impact Analysis by Mining Software and Change Request Repositories. In *Proc. of the 11th International Symposium on Software Metrics*, pages 9–29, 2005.

[92] G. Canfora and L. Cerulo. Fine Grained Indexing of Software Repositories to Support Impact Analysis. In *Proc. of the International Workshop on Mining Software Repositories*, pages 105–111, 2006.

[93] G. Canfora and L. Cerulo. Supporting Change Request Assignment in Open Source Development. In *Proc. of the 21st Symposium on Applied Computing*, pages 1767–1772, 2006.

[94] L. Cao and B. Ramesh. Agile Requirements Engineering Practices: An Empirical Study. *IEEE Software*, 25(1):60–67, 2008.

[95] G. Capobianco, A. De Lucia, R. Oliveto, A. Panichella, and S. Panichella. On the Role of the Nouns in IR-based Traceability Recovery. In *Proc. of the 17th International Conference on Program Comprehension*, pages 148–157, 2009.

[96] G. Capobianco, A. De Lucia, R. Oliveto, A. Panichella, and S. Panichella. Traceability Recovery Using Numerical Analysis. In *Proc. of the 16th Working Conference on Reverse Engineering*, pages 195–204, 2009.

[97] Carnegie Mellon Software Engineering Institute. *CMMI for Development, Version 1.3*. 2010.

[98] A. Casamayor, D. Godoy, and M. Campo. Identification of Non-Functional Requirements in Textual Specifications: A Semi-Supervised Learning Approach. *Information and Software Technology*, 52(4):436–445, 2010.

[99] N. Castell, O. Slavkova, Y. Toussaint, and A. Tuells. Quality Control of Software Specifications Written in Natural Language. In *Proc. of the 7th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pages 37–44, 1994.

[100] Y. Cavalcanti, P. Silveira Neto, I. Machado, T. Vale, E. Almeida, and S. Meira. Challenges and Opportunities for Software Change Request Repositories: A Systematic Mapping Study. *Journal of Software: Evolution and Process*, 26(7):620–653, 2014.

[101] J. Chang and D. Blei. Hierarchical Relational Models for Document Networks. *The Annals of Applied Statistics*, 4(1):124–150, 2010.

[102] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental Clustering and Dynamic Information Retrieval. In *Proc. of the 29th Annual ACM Symposium on Theory of Computing*, pages 626–635, 1997.

[103] K. Chen and V. Rajlich. RIPPLES: Tool for Change in Legacy Software. In *Proc. of the 17th International Conference on Software Maintenance*, pages 230–239, 2001.

[104] L. Chen, X. Wang, and C. Liu. An Approach to Improving Bug Assignment with Bug Tossing Graphs and Bug Similarities. *Journal of Software*, 6(3):421–427, 2011.

[105] X. Chen. Extraction and Visualization of Traceability Relationships between Documents and Source Code. In *Proc. of the 25th International Conference on Automated Software Engineering*, pages 505–509, 2010.

[106] X. Chen and J. Grundy. Improving Automated Documentation to Code Traceability by Combining Retrieval Techniques. In *Proc. of the 26th International Conference on Automated Software Engineering*, pages 223–232, 2011.

[107] X. Chen, J. Hosking, and J. Grundy. A Combination Approach for Enhancing Automated Traceability. In *Proc. of the 33rd International Conference on Software Engineering*, pages 912–915, 2011.

[108] B. Cheng and J. Atlee. Research Directions in Requirements Engineering. In *Proc. of the Future of Software Engineering*, pages 285–303, 2007.

[109] I. Chou. Secure Software Configuration Management Processes for Nuclear Safety Software Development Environment. *Annals of Nuclear Energy*, 38(10):2174–2179, 2011.

[110] J. Cleland-Huang. Traceability in Agile Projects. In J. Cleland-Huang, O. Gotel, and A. Zisman, editors, *Software and Systems Traceability*, pages 265–275. Springer, 2012.

[111] J. Cleland-Huang, C. Chang, and M. Christensen. Event-Based Traceability for Managing Evolutionary Change. *Transactions on Software Engineering*, 29(9):796–810, 2003.

[112] J. Cleland-Huang, A. Czauderna, A. Dekhtyar, O. Gotel, J. Huffman Hayes, E. Keenan, J. Maletic, D. Poshyvanyk, Y. Shin, A. Zisman, G. Antoniol, B. Berenbach, A. Egyed, and P. Mäder. Grand Challenges, Benchmarks, and TraceLab: Developing Infrastructure for the Software Traceability Research Community. In *Proc. of the 6th International Workshop on Traceability in Emerging Forms of Software Engineering*, pages 17–23, 2011.

[113] J. Cleland-Huang, O. Gotel, and A. Zisman, editors. *Software and Systems Traceability*. Springer, 2012.

[114] J. Cleland-Huang and J. Guo. Towards More Intelligent Trace Retrieval Algorithms. In *Proc. of the 3rd Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*, 2014.

[115] J. Cleland-Huang, M. Heimdahl, J. Huffman Hayes, R. Lutz, and P. Mäder. Trace Queries for Safety Requirements in High Assurance Systems. In *Proc. of the 18th International Working Conference Requirements Engineering: Foundation for Software Quality*, pages 179–193, 2012.

[116] J. Cleland-Huang, J. Huffman Hayes, and A. Dekhtyar. Center of Excellence for Traceability: Problem Statement and Grand Challenges in Traceability (v0.1). Technical Report COET-GCT-06-01-0.9, 2006.

[117] J. Cleland-Huang, W. Marrero, and B. Berenbach. Goal-Centric Traceability: Using Virtual Plumblines to Maintain Critical Systemic Qualities. *Transactions on Software Engineering*, 34(5):685–699, 2008.

[118] J. Cleland-Huang, R. Settimi, C. Duan, and X. Zou. Utilizing Supporting Evidence to Improve Dynamic Requirements Traceability. In *Proc. of the 13th International Conference on Requirements Engineering*, pages 135–144, 2005.

[119] J. Cleland-Huang, Y. Shin, E. Keenan, A. Czauderna, G. Leach, E. Moritz, M. Gethers, D. Poshyvanyk, J. Huffman Hayes, and W. Li. Toward Actionable, Broadly Accessible Contests in Software Engineering. In *Proc. of the 34th International Conference on Software Engineering*, pages 1329–1332, 2012.

[120] C. Cleverdon. The Significance of the Cranfield Tests on Index Languages. In *Proc. of the 14th Annual International SIGIR Conference on Research and Development in Information Retrieval*, pages 3–12, 1991.

[121] R. Cooper. Stage-Gate Systems: A New Tool for Managing New Products. *Business Horizons*, 33(3):44–54, 1990.

[122] B. Croft, H. Turtle, and D. Lewis. The Use of Phrases and Structured Queries in Information Retrieval. In *Proc. of the 14th Annual International Conference on Research and Development in Information Retrieval*, pages 32–45, 1991.

[123] D. Cruzes and T. Dybå. Recommended Steps for Thematic Synthesis in Software Engineering. In *Proc. of the 5th International Symposium on Empirical Software Engineering and Measurement*, pages 275–284, 2011.

[124] D. Cruzes and T. Dybå. Research Synthesis in Software Engineering: A Tertiary Study. *Information and Software Technology*, 53(5):440–455, 2011.

[125] D. Cubranić. Automatic Bug Triage Using Text Categorization. In *Proc. of the 16th International Conference on Software Engineering & Knowledge Engineering*, pages 92–97, 2004.

[126] D. Cubranić. *Project History as a Group Memory: Learning From the Past*. PhD Thesis, University of British Columbia, 2004.

[127] D. Cubranić, G. Murphy, J. Singer, and K. Booth. Hipikat: A Project Memory for Software Development. *Transactions on Software Engineering*, 31(6):446–465, 2005.

[128] D. Cuddeback, A. Dekhtyar, and J. Huffman Hayes. Automated Requirements Traceability: The Study of Human Analysts. In *Proc. of the 18th International Requirements Engineering Conference*, pages 231–240, 2010.

[129] G. Cumming. *Understanding The New Statistics: Effect Sizes, Confidence Intervals, and Meta-Analysis*. Routledge, 2012.

[130] L. Da Costa and M. Schoenauer. Bringing Evolutionary Computation to Industrial Applications with GUIDE. In *Proc. of the 11th Annual Conference on Genetic and Evolutionary Computation*, pages 1467–1474, 2009.

[131] F. da Silva, A. Santos, S. Soares, C. Franca, C. Monteiro, and F. Maciel. Six Years of Systematic Literature Reviews in Software Engineering: An Updated Tertiary Study. *Information and Software Technology*, 53(9):899–913, 2011.

[132] B. Dagenais, H. Ossher, R. Bellamy, M. Robillard, and J. de Vries. Moving Into A New Software Project Landscape. In *Proc. of the 32nd International Conference on Software Engineering*, pages 275–284, 2010.

[133] D. Damian and J. Chisan. An Empirical Study of the Complex Relationships between Requirements Engineering Processes and Other Processes that Lead to Payoffs in Productivity, Quality, and Risk Management. *Transactions on Software Engineering and Methodology*, 32(7):433–453, 2006.

[134] D. Damian, J. Chisan, L. Vaidyanathasamy, and Y. Pal. Requirements Engineering and Downstream Software Development: Findings from a Case Study. *Empirical Software Engineering*, 10(3):255–283, 2005.

[135] C. Dantas, L. Murta, and C. Werner. Mining Change Traces from Versioned UML Repositories. In *Proc. of the Brazilian Symposium of Software Engineering*, pages 236–252, 2007.

[136] J. de la Vara, M. Borg, K. Wnuk, and L. Moonen. Survey on Safety Evidence Change Impact Analysis in Practice: Detailed Description and Analysis. Technical Report 18, Simula Research Laboratory, 2014.

[137] A. De Lucia, M. Di Penta, and R. Oliveto. Improving Source Code Lexicon via Traceability and Information Retrieval. *Transactions on Software Engineering*, 37(2):205–227, 2011.

[138] A. De Lucia, M. Di Penta, R. Oliveto, and F. Zurolo. COCONUT: COde COmprehension Nurturant Using Traceability. In *Proc. of the 22nd International Conference on Software Maintenance*, pages 274–275, 2006.

[139] A. De Lucia, M. Di Penta, R. Oliveto, and F. Zurolo. Improving Comprehensibility of Source Code via Traceability Information: A Controlled Experiment. In *Proc. of the International Conference on Program Comprehension*, pages 317–326, 2006.

[140] A. De Lucia, F. Fasano, and R. Oliveto. Traceability Management for Impact Analysis. In *Proc. of the Frontiers of Software Maintenance*, pages 21–30, 2008.

[141] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora. Enhancing an Artefact Management System with Traceability Recovery Features. In *Proc. of the 20th International Conference on Software Maintenance*, pages 306–315, 2004.

[142] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora. ADAMS Re-Trace: A Traceability Recovery Tool. In *Proc. of the 9th European Conference on Software Maintenance and Reengineering*, pages 32–41, 2005.

[143] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora. Can Information Retrieval Techniques Effectively Support Traceability Link Recovery? In *Proc. of the 14th International Conference on Program Comprehension*, pages 307–316, 2006.

[144] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora. Recovering Traceability Links in Software Artifact Management Systems Using Information Retrieval Methods. *Transactions on Software Engineering and Methodology*, 16(4:13), 2007.

[145] A. De Lucia, A. Marcus, R. Oliveto, and D. Poshyvanyk. Information Retrieval Methods for Automated Traceability Recovery. In J. Cleland-Huang, O. Gotel, and A. Zisman, editors, *Software and Systems Traceability*. Springer, 2012.

[146] A. De Lucia, R. Oliveto, and P. Sgueglia. Incremental Approach and User Feedbacks: A Silver Bullet for Traceability Recovery? In *Proc. of the 22nd International Conference on Software Maintenance*, pages 299–308, 2006.

[147] A. De Lucia, R. Oliveto, and G. Tortora. IR-based Traceability Recovery Processes: An Empirical Comparison of "One-Shot" and Incremental Processes. In *Proc. of the 23rd International Conference on Automated Software Engineering*, pages 39–48, 2008.

[148] A. De Lucia, R. Oliveto, and G. Tortora. Assessing IR-Based Traceability Recovery Tools Through Controlled Experiments. *Empirical Software Engineering*, 14(1):57–92, 2009.

[149] A. De Lucia, R. Oliveto, and G. Tortora. The Role of the Coverage Analysis during IR-based Traceability Recovery: A Controlled Experiment. In *Proc. of the 25th International Conference on Software Maintenance*, pages 371–380, 2009.

[150] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman. Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

[151] A. Dekhtyar, O. Dekhtyar, J. Holden, J. Huffman Hayes, D. Cuddeback, and W. Kong. On Human Analyst Performance in Assisted Requirements Tracing: Statistical Analysis. In *Proc. of the 19th International Requirements Engineering Conference*, pages 111–120, 2011.

[152] A. Dekhtyar and J. Huffman Hayes. Good Benchmarks are Hard To Find: Toward the Benchmark for Information Retrieval Applications in Software Engineering. *Proc. of the 22nd International Conference on Software Maintenance*, 2006.

[153] A. Dekhtyar, J. Huffman Hayes, and G. Antoniol. Benchmarks for Traceability? In *Proc. of the International Symposium on Grand Challenges in Traceability*, 2007.

[154] A. Dekhtyar, J. Huffman Hayes, and J. Larsen. Make the Most of Your Time: How Should the Analyst Work with Automated Traceability Tools? In *Proc. of the 3rd International Workshop on Predictor Models in Software Engineering*, page 4, 2007.

[155] A. Dekhtyar, J. Huffman Hayes, S. Sundaram, A. Holbrook, and O. Dekhtyar. Technique Integration for Requirements Assessment. In *Proc. of the 15th International Requirements Engineering Conference*, pages 141–152, 2007.

[156] C. Dekkers and P. McQuaid. The Dangers of Using Software Metrics to (Mis) Manage. *IT Professional*, 4(2):24–30, 2002.

[157] F. Di and M. Zhang. An Improving Approach for Recovering Requirements-to-Design Traceability Links. In *Proc. of the 1stInternational Conference on Computational Intelligence and Software Engineering*, pages 1–6, 2009.

[158] A. Dias Neto, R. Subramanyan, M. Vieira, and G. Travassos. A Survey on Model-based Testing Approaches: A Systematic Review. In *Proc. of the 1st International Workshop on Empirical Assessment of Software Engineering Languages and Technologies*, pages 31–36, 2007.

[159] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk. Feature Location in Source Code: A Taxonomy and Survey. *Journal of Software Maintenance and Evolution: Research and Practice*, 25(1):53–95, 2011.

[160] R. Dömges and K. Pohl. Adapting Traceability Environments to Project-Specific Needs. *Communications of the ACM*, 41(12):54–62, 1998.

[161] P. Domingos. A Few Useful Things to Know About Machine Learning. *Communications of the ACM*, 55(10):78–87, 2012.

[162] M. Dorfman. *Standards, Guidelines, and Examples on System and Software Requirements Engineering*. IEEE Computer Society Press, 1994.

[163] C. Duan and J. Cleland-Huang. Clustering Support for Automated Tracing. In *Proc. of the 22nd International Conference on Automated Software Engineering*, pages 244–253, 2007.

[164] A. Dubey and J. Hudepohl. Towards Global Deployment of Software Engineering Tools. In *Proc. of the 8th International Conference on Global Software Engineering*, pages 129–133, 2013.

[165] K. Dunn. Design and Analysis of Experiments. In *Process Improvement Using Data*, pages 207–288. 294-34b8 edition, 2014.

[166] T. Dybå and T. Dingsøyr. Strength of Evidence in Systematic Reviews in Software Engineering. In *Proc. of the 2nd International Symposium on Empirical Software Engineering and Measurement*, pages 178–187, 2008.

[167] S. Easterbrook, J. Singer, M. Storey, and D. Damian. Selecting Empirical Methods for Software Engineering Research. In F. Shull, J. Singer, and D. Sjøberg, editors, *Guide to Advanced Empirical Software Engineering*, pages 285–311. Springer, 2008.

[168] A. Egyed and P. Grünbacher. Automating Requirements Traceability: Beyond the Record Replay Paradigm. In *Proc. of the 17th International Conference on Automated Software Engineering*, pages 163–171, 2002.

[169] T. Eisenbarth, R. Koschke, and D. Simon. Locating Features in Source Code. *Transactions on Software Engineering*, 29(3):210–224, 2003.

[170] S. Eldh, J. Brandt, M. Street, H. Hansson, and S. Punnekkat. Towards Fully Automated Test Management for Large Complex Systems. In *Proc. of the 3rd International Conference on Software Testing, Verification and Validation*, pages 412–420, 2010.

[171] E. Engström, P. Runeson, and M. Skoglund. A Systematic Review on Regression Test Selection Techniques. *Information and Software Technology*, 52(1):14–30, 2010.

[172] M. Eppler and J. Mengis. The Concept of Information Overload: A Review of Literature from Organization Science, Accounting, Marketing, MIS, and Related Disciplines. *The Information Society*, 20(5):325–344, 2004.

[173] J. Estublier. Software Configuration Management: A Roadmap. In *Proc. of the Conference on The Future of Software Engineering*, pages 279–289, 2000.

[174] European Committee for Electrotechnical Standardisation. *Railway Applications - Safety Related Electronic Systems for Signaling*. 1999.

[175] D. Falessi, G. Cantone, and G. Canfora. A Comprehensive Characterization of NLP Techniques for Identifying Equivalent Requirements. In *Proc. of the 4th International Symposium on Empirical Software Engineering and Measurement*, pages 100–110, 2010.

[176] K. Fang, R. Li, and A. Sudjianto. *Design and Modeling for Computer Experiments*. CRC Press, 2006.

[177] R. Feldt. Do System Test Cases Grow Old? In *Proc. of the 7th. International Conference on Software Testing, Verification and Validation*, pages 343–352, 2014.

[178] R. Feldt and P. Nordin. Using Factorial Experiments to Evaluate the Effect of Genetic Programming Parameters. In *Proc. of the 3rd European Conference on Genetic Programming*, pages 271–282, 2000.

[179] A. Felfernig, M. Jeran, G. Ninaus, F. Reinfrank, S. Reiterer, and M. Stettinger. Basic Approaches in Recommendation Systems. In M. Robillard, W. Maalej, R. Walker, and T. Zimmermann, editors, *Recommendation Systems in Software Engineering*, pages 15–37. Springer, 2014.

[180] K. Felizardo, N. Salleh, R. Martins, E. Mendes, S. MacDonell, and J. Maldonado. Using Visual Text Mining to Support the Study Selection Activity in Systematic Literature Reviews. In *Proc. of the 5th International Symposium on Empirical Software Engineering and Measurement*, pages 77–86, 2011.

[181] R. Ferguson and G. Lami. An Empirical Study on the Relationship between Defective Requirements and Test Failures. In *Proc. of the 30th Software Engineering Workshop*, pages 7–10, 2006.

[182] A. Fisher. *CASE: Using Software Development Tools*. Wiley, 2nd edition, 1991.

[183] R. Fiutem and G. Antoniol. Identifying Design-Code Inconsistencies in Object-Oriented Software: A Case Study. In *Proc. of the 14th International Conference on Software Maintenance*, pages 94–102, 1998.

[184] N. Fogelström and T. Gorschek. Test-Case Driven Versus Checklist-Based Inspections of Software Requirements - An Experimental Evaluation. In *Proc. of the 10th Workshop on Requirements Engineering*. 2007.

[185] E. Frank, M. Hall, G. Holmes, R. Kirkby, B. Pfahringer, I. Witten, and L. Trigg. Weka - A Machine Learning Workbench for Data Mining. In O. Maimon and L. Rokach, editors, *Data Mining and Knowledge Discovery Handbook*, pages 1305–1314. Springer, 2005.

[186] E. Frank, M. Hall, L. Trigg, G. Holmes, and I. Witten. Data Mining in Bioinformatics Using Weka. *Bioinformatics*, 20(15):2479–2481, 2004.

[187] G. Fraser and A. Arcuri. EvoSuite: Automatic Test Suite Generation for Object-oriented Software. In *Proc. of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, pages 416–419, 2011.

[188] G. Fraser and A. Arcuri. The Seed is Strong: Seeding Strategies in Search-Based Software Testing. In *Proc. of the 5th International Conference on Software Testing, Verification and Validation*, pages 121–130, 2012.

[189] L. Freund, E. Toms, and J. Waterhouse. Modeling the Information Behaviour of Software Engineers Using a Work-Task Framework. *Proceedings of the American Society for Information Science and Technology*, 42(1), 2005.

[190] Y. Freund and R. Schapire. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[191] G. Gay, S. Haiduc, A. Marcus, and T. Menzies. On the Use of Relevance Feedback in IR-based Concept Location. In *Proc. of the 25th International Conference on Software Maintenance*, pages 351–360, 2009.

[192] M. Gethers, B. Dit, H. Kagdi, and D. Poshyvanyk. Integrated Impact Analysis for Managing Software Changes. In *Proc. of the 34th International Conference on Software Engineering*, pages 430–440, 2012.

[193] M. Gethers, H. Kagdi, B. Dit, and D. Poshyvanyk. An Adaptive Approach to Impact Analysis from Change Requests to Source Code. In *Proc. of the 26th International Conference on Automated Software Engineering*, pages 540–543, 2011.

[194] M. Gethers, R. Oliveto, D. Poshyvanyk, and A. De Lucia. On Integrating Orthogonal Information Retrieval Methods to Improve Traceability Recovery. In *Proc. of the 27th International Conference on Software Maintenance*, pages 133–142, 2011.

[195] L. Getoor and C. Diehl. Link Mining: A Survey. *SIGKDD Explorations Newsletter*, 7(2):3–12, 2005.

[196] S. Ghosh, S. Ramaswamy, and R. Jetley. Towards Requirements Change Decision Support. In *Proc. of the 20th Asia-Pacific Software Engineering Conference*, pages 148–155, 2013.

[197] M. Gibiec, A. Czauderna, and J. Cleland-Huang. Towards Mining Replacement Queries for Hard-To-Retrieve Traces. In *Proc. of the 25th International Conference on Automated Software Engineering*, pages 245–254, 2010.

[198] T. Gorschek and A. Davis. Requirements Engineering: In Search of the Dependent Variables. *Information and Software Technology*, 50(1-2):67–75, 2008.

[199] T. Gorschek and C. Wohlin. Packaging Software Process Improvement Issues - A Method and A Case Study. *Software: Practice & Experience*, 34(14):1311–1344, 2004.

[200] T. Gorschek and C. Wohlin. Requirements Abstraction Model. *Requirements Engineering*, 11(1):79–101, 2006.

[201] T. Gorschek, C. Wohlin, P. Carre, and S. Larsson. A Model for Technology Transfer in Practice. *IEEE Software*, 23(6):88–95, 2006.

[202] O. Gotel, J. Cleland-Huang, J. Huffman Hayes, A. Zisman, A. Egyed, P. Grünbacher, A. Dekhtyar, G. Antoniol, and J. Maletic. The Grand Challenge of Traceability (v1.0). In J. Cleland-Huang, O. Gotel, and A. Zisman, editors, *Software and Systems Traceability*, pages 343–409. Springer, 2012.

[203] O. Gotel, J. Cleland-Huang, J. Huffman Hayes, A. Zisman, A. Egyed, P. Grünbacher, A. Dekhtyar, G. Antoniol, J. Maletic, and P. Mäder. Traceability Fundamentals. In J. Cleland-Huang, O. Gotel, and A. Zisman, editors, *Software and Systems Traceability*, pages 3–22. Springer, 2012.

[204] O. Gotel and C. Finkelstein. An Analysis of the Requirements Traceability Problem. In *Proc. of the 1st International Conference on Requirements Engineering*, pages 94–101, 1994.

[205] D. Graham. Requirements and Testing: Seven Missing-link Myths. *IEEE Software*, 19(5):15–17, 2002.

[206] S. Green. How Many Subjects Does It Take To Do A Regression Analysis. *Multivariate Behavioral Research*, 26(3):499–510, 1991.

[207] W. Grieskamp, N. Kicillof, K. Stobie, and V. Braberman. Model-based Quality Assurance of Protocol Documentation: Tools and Methodology. *Software Testing, Verification and Reliability*, 21(1):55–71, 2011.

[208] K. Grimm. Software Technology in an Automotive Company: Major Challenges. In *Proc. of the 25th International Conference on Software Engineering*, pages 498–503, 2003.

[209] E. Gummesson. *Qualitative Methods in Management Research*. SAGE Publications, 1999.

[210] I. Habli, R. Hawkins, and T. Kelly. Software Safety: Relating Software Assurance and Software Integrity. *International Journal of Critical Computer-Based Systems*, 1(4):364–383, 2010.

[211] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten. The WEKA Data Mining Software: An Update. *SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[212] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell. A Systematic Literature Review on Fault Prediction Performance in Software Engineering. *Transactions on Software Engineering*, 38(6):1276–1304, 2012.

[213] M. Harman. The Current State and Future of Search Based Software Engineering. In *Proc. of the Future of Software Engineering*, pages 342–357, 2007.

[214] B. Hasling, H. Götz, and K. Beetz. Model Based Testing of System Requirements using UML Use Case Models. In *Proc. of the 1st International Conference on Software Testing, Verification, and Validation*, pages 367–376, 2008.

[215] E. Hatcher and O. Gospodnetic. *Lucene in Action*. Manning Publications, 2004.

[216] M. Heindl and S. Biffl. A Case Study on Value-Based Requirements Tracing. In *Proc. of the 10th European Software Engineering Conference*, pages 60–69, 2005.

[217] J. Helming, H. Arndt, Z. Hodaie, M. Koegel, and N. Narayan. Automatic Assignment of Work Items. In *Proc. of the 5th International Conference on Evaluation of Novel Approaches to Software Engineering*, pages 236–250, 2011.

[218] K. Herzig and A. Zeller. Mining Bug Data. In M. Robillard, W. Maalej, R. Walker, and T. Zimmermann, editors, *Recommendation Systems in Software Engineering*, pages 131–171. Springer, 2014.

[219] A. Hevner, S. March, J. Park, and S. Ram. Design Science in Information Systems Research. *MIS Quarterly*, 28(1):75–105, 2004.

[220] Hewlett Packard Development Company. HP Quality Center Software. Data sheet 4AA0-9587ENW rev. 3, 2009.

[221] T. Hofman. Unsupervised Learning by Probabilistic Latent Semantic Analysis. *Machine Learning*, 42(1-2):177–196, 2001.

[222] M. Hofmann and R. Klinkenberg, editors. *RapidMiner: Data Mining Use Cases and Business Analytics Applications*. Chapman & Hall/CRC Press, 2013.

[223] D. Hopwood. Forty Years of Genetics with Streptomyces: From In Vivo Through In Vitro to In Silico. *Microbiology*, 145(9):2183–2202, 1999.

[224] M. Höst, R. Feldt, and F. Luders. Support for Different Roles in Software Engineering Master's Thesis Projects. *Transactions on Education*, 53(2):288–296, 2010.

[225] M. Höst, B. Regnell, and C. Wohlin. Using Students as Subjects - A Comparative Study of Students and Professionals in Lead-Time Impact Assessment. *Empirical Software Engineering*, 5(3):201–214, 2000.

[226] G. Huang, D. Wang, and Y. Lan. Extreme Learning Machines: A Survey. *International Journal of Machine Learning and Cybernetics*, 2(2):107–122, 2011.

[227] J. Huang, R. White, and S. Dumais. No Clicks, No Problem: Using Cursor Movements to Understand and Improve Search. In *Proc. of the 29th Conference on Human Factors in Computing Systems*, pages 1225–1234, 2011.

[228] J. Huffman Hayes, G. Antoniol, and Y. Guéhéneuc. PREREQIR: Recovering Pre-Requirements via Cluster Analysis. In *Proc. of the 15th Working Conference on Reverse Engineering*, pages 165–174, 2008.

[229] J. Huffman Hayes and A. Dekhtyar. A Framework for Comparing Requirements Tracing Experiments. *International Journal of Software Engineering and Knowledge Engineering*, 15(5):751–781, 2005.

[230] J. Huffman Hayes and A. Dekhtyar. Humans in the Traceability Loop: Can't Live With 'em, Can't Live Without 'em. In *Proc. of the 3rd International Workshop on Traceability in Emerging Forms of Software Engineering*, pages 20–23, 2005.

[231] J. Huffman Hayes, A. Dekhtyar, and S. Sundaram. Advancing Candidate Link Generation for Requirements Tracing: The Study of Methods. *Transactions on Software Engineering*, 32(1):4–19, 2006.

[232] J. Huffman Hayes, A. Dekhtyar, S. Sundaram, A. Holbrook, S. Vadlamudi, and A. April. REquirements TRacing On target (RETRO): Improving software maintenance through traceability recovery. *Innovations in Systems and Software Engineering*, 3(3):193–202, 2007.

[233] J. Huffman Hayes, A. Dekhtyar, S. Sundaram, and S. Howard. Helping Analysts Trace Requirements: An Objective Look. In *Proc. of the 12th International Conference on Requirements Engineering*, pages 249–259, 2004.

[234] J. Huffman Hayes, H. Sultanov, W. Kong, and W. Li. Software Verification and Validation Research Laboratory (SVVRL) of the University of Kentucky: Traceability Challenge 2011: Language Translation. In *Proc. of the 6th International Workshop on Traceability in Emerging Forms of Software Engineering*, pages 50–53, 2011.

[235] IEEE Computer Society. 610.12-1990 IEEE Standard Glossary of Software Engineering Terminology. Technical report, 1990.

[236] L. Ilzarbe, M. Álvarez, E. Viles, and M. Tanco. Practical Applications of Design of Experiments in the Field of Engineering: A Bibliographical Review. *Quality and Reliability Engineering International*, 24(4):417–428, 2008.

[237] P. Ingwersen and K. Järvelin. *The Turn: Integration of Information Seeking and Retrieval in Context*. Springer, 2005.

[238] International Electrotechnical Commission. *IEC 61511-1 ed 1.0, Safety Instrumented Systems for the Process Industry Sector*. 2003.

[239] International Electrotechnical Commission. *IEC 61508 ed 1.0, Electrical/-Electronic/Programmable Electronic Safety-Related Systems*. 2010.

[240] International Electrotechnical Commission. *IEC 61131-3 ed 3.0, Programmable Controllers - Part 3: Programming Languages*. 2013.

[241] International Organization for Standardization. ISO/IEC 9126-1:2001(E) International Standard Software Engineering Product Quality Part 1: Quality Model. Technical report, ISO/IEC, 2001.

[242] International Organization for Standardization. *ISO 26262-1:2011 Road Vehicles - Functional Safety*. 2011.

[243] N. Jalbert and W. Weimer. Automated Duplicate Detection for Bug Tracking Systems. In *Proc. of the 38th International Conference on Dependable Systems and Networks*, pages 52–61, 2008.

[244] B. Jansen. Search Log Analysis: What It Is, What's Been Done, How To Do It. *Library & Information Science Research*, 28(3):407–432, 2006.

[245] M. Jarke. Requirements Tracing. *Communications of the ACM*, 41(12):32–36, 1998.

[246] K. Järvelin and J. Kekälainen. IR Evaluation Methods for Retrieving Highly Relevant Documents. In *Proc. of the 23rd Annual International SIGIR Conference on Research and Development in Information Retrieval*, pages 41–48, 2000.

[247] A. Jedlitschka, M. Ciolkowski, and D. Pfahl. Reporting Experiments in Software Engineering. In F. Shull, J. Singer, and D. Sjøberg, editors, *Guide to Advanced Empirical Software Engineering*, pages 201–228. Springer, 2008.

[248] G. Jeong, S. Kim, and T. Zimmermann. Improving Bug Triage with Bug Tossing Graphs. In *Proc. of the the 7th Joint Meeting of the European Software Engineering Conference and the Symposium on The Foundations of Software Engineering*, pages 111–120, 2009.

[249] H. Jiang, T. Nguyen, I. Chen, H. Jaygarl, and C. Chang. Incremental Latent Semantic Indexing for Automatic Traceability Link Evolution Management. In *Proc. of the 23rd International Conference on Automated Software Engineering*, pages 59–68, 2008.

[250] J. Johansson. *Beyond Textual Information in Defect Duplicate Detection: An Exploratory Study in the Android Issue Tracker*. MSc Thesis, Lund University, http://sam.cs.lth.se/ExjobGetFile?id=628, 2014.

[251] D. Jones. A Taxonomy of Global Optimization Methods Based on Response Surfaces. *Journal of Global Optimization*, 21(4):345–383, 2001.

[252] J. Jones, M. Grechanik, and A. van der Hoek. Enabling and Enhancing Collaborations between Software Development Organizations and Independent Test Agencies. In *Proc. of the 2nd Workshop on Cooperative and Human Aspects on Software Engineering*, pages 56–59, 2009.

[253] H. Jonsson, S. Larsson, and S. Punnekkat. Agile Practices in Regulated Railway Software Development. In *Proc. of the 23rd International Symposium on Software Reliability Engineering Workshops*, pages 355–360, 2012.

[254] L. Jonsson, M. Borg, D. Broman, K. Sandahl, S. Eldh, and P. Runeson. Automated Bug Assignment: Ensemble-based Machine Learning in Large Scale Industrial Contexts. *Under revision in Empirical Software Engineering*, 2015.

[255] L. Jonsson, D. Broman, K. Sandahl, and S. Eldh. Towards Automated Anomaly Report Assignment in Large Complex Systems Using Stacked Generalization. In *Proc. of the 5th International Conference on Software Testing, Verification and Validation*, pages 437–446, 2012.

[256] Z. Jourdan, K. Rainer, and T. Marshall. Business Intelligence: An Analysis of the Literature. *Information Systems Management*, 25(2):121–131, 2008.

[257] S. Just, R. Premraj, and T. Zimmermann. Towards the Next Generation of Bug Tracking Systems. In *Proc. of the 24th Symposium on Visual Languages and Human-Centric Computing*, pages 82–85, 2008.

[258] H. Kagdi, M. Collard, and J. 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.

[259] H. Kagdi, M. Gethers, D. Poshyvanyk, and M. Collard. Blending Conceptual and Evolutionary Couplings to Support Change Impact Analysis in Source Code. In *Proc. of the 17th Working Conference on Reverse Engineering*, pages 119–128, 2010.

[260] H. Kagdi, M. Gethers, D. Poshyvanyk, and M. Hammad. Assigning Change Requests to Software Developers. *Journal of Software: Evolution and Process*, 24(1):3–33, 2012.

[261] H. Kagdi, J. Maletic, and B. Sharif. Mining Software Repositories for Traceability Links. In *Proc. of the 15th International Conference on Program Comprehension*, pages 145–154, 2007.

[262] C. Kaner and W. Bond. Software Engineering Metrics: What Do They Measure and How Do We Know? In *Proc. of 10th International Symposium on Software Metrics*, 2004.

[263] P. Karr-Wisniewski and Y. Lu. When More is Too Much: Operationalizing Technology Overload and Exploring Its Impact on Knowledge Worker Productivity. *Computers in Human Behavior*, 26(5):1061–1072, 2010.

[264] V. Katta and T. Stålhane. A Conceptual Model of Traceability for Safety Systems. In *Proc. of the 2nd Conference on Complex Systems Design & Management*, pages 1–12, 2011.

[265] N. Kaushik, L. Tahvildari, and M. Moore. Reconstructing Traceability Between Bugs and Test Cases: An Experimental Study. In *Proc. of the 20th Working Conference on Reverse Engineering*, pages 411–414, 2011.

[266] E. Keenan, A. Czauderna, G. Leach, J. Cleland-Huang, Y. Shin, E. Moritz, M. Gethers, D. Poshyvanyk, J. Maletic, J. Huffman Hayes, A. Dekhtyar, D. Manukian, S. Hossein, and D. Hearn. TraceLab: An Experimental Workbench for Equipping Researchers to Innovate, Synthesize, and Comparatively Evaluate Traceability Solutions. In *Proc. of the 34th International Conference on Software Engineering*, pages 1375–1378, 2012.

[267] J. Kekälainen and K. Järvelin. Evaluating Information Retrieval Systems Under the Challenges of Interaction and Multidimensional Dynamic Relevance. In *Proc. of the 4th CoLIS Conference*, pages 253–270, 2002.

[268] T. Kelly. *Arguing Safety - A Systematic Approach to Managing Safety Cases*. PhD Thesis, University of York, 1999.

[269] M. Kersten and G. Murphy. Using Task Context to Improve Programmer Productivity. In *Proc. of the 14th International Symposium on Foundations of Software Engineering*, pages 1–11, 2006.

[270] M. Kilpinen. *The Emergence of Change at the Systems Engineering and Software Design Interface*. PhD thesis, University of Cambridge, 2008.

[271] M. Kilpinen, C. Eckert, and P. Clarkson. Assessing Impact Analysis Practice to Improve Change Management Capability. In *Proc. of the 17th International Conference on Engineering Design*, pages 205–216, 2009.

[272] B. Kitchenham, D. Budgen, and P. Brereton. Using Mapping Studies as the Basis for Further Research - A Participant-Observer Case Study. *Information and Software Technology*, 53(6):638–651, 2011.

[273] B. Kitchenham and S. Charters. Guidelines for Performing Systematic Literature Reviews in Software Engineering. *EBSE Technical Report*, 2007.

[274] B. Kitchenham, T. Dybå, and M. Jørgensen. Evidence-based Software Engineering. In *Proc. of the 26th International Conference on Software Engineering*, pages 273–281, 2004.

[275] B. Kitchenham, S. Pfleeger, L Pickard, P. Jones, D. Hoaglin, K. El Emam, and J. Rosenberg. Preliminary Guidelines for Empirical Research in Software Engineering. *Transactions on Software Engineering and Methodology*, 28(8):721–734, 2002.

[276] J. Kleijnen. Low-Order Polynomial Regression Metamodels and Their Designs: Basics. In *Design and Analysis of Simulation Experiments*, pages 15–71. Springer, 2008.

[277] J. Kleijnen. Screening Designs. In *Design and Analysis of Simulation Experiments*, pages 157–173. Springer, 2008.

[278] H. Klein and M. Myers. A Set of Principles for Conducting and Evaluating Interpretive Field Studies in Information Systems. *MIS Quarterly*, 23(1):67–93, 1999.

[279] A. Klevin. *People, Process and Tools: A Study of Impact Analysis in a Change Process*. Master Thesis, Lund University, http://sam.cs.lth.se/ExjobGetFile?id=434, 2012.

[280] J. Kodovsky. On Dangers of Cross-Validation in Steganalysis. Technical report, Birmingham University, 2011.

[281] R. Kohavi. A Study of Cross-validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proc. of the 14th International Joint Conference on Artificial Intelligence*, pages 1137–1143, 1995.

[282] L. Kong, J. Li, Y. Li, Y. Yang, and Q. Wang. A Requirement Traceability Refinement Method Based on Relevance Feedback. In *Proc. of the 21st International Conference on Software Engineering and Knowledge Engineering*, pages 37–42, 2009.

[283] R. Kraut and L. Streeter. Coordination in Software Development. *Communications of the ACM*, 38(3):69–81, 1995.

[284] P. Kruchten. *The Rational Unified Process: An Introduction*. Addison-Wesley Professional, 2004.

[285] J. Kruschke. *Doing Bayesian Data Analysis: A Tutorial Introduction with R*. Academic Press, 2010.

[286] J. Kukkanen, K. Vakevainen, M. Kauppinen, and E. Uusitalo. Applying a Systematic Approach to Link Requirements and Testing: A Case Study. In *Proc. of the 16th Asia-Pacific Software Engineering Conference*, pages 482–488, 2009.

[287] L. Kuncheva and C. Whitaker. Measures of Diversity in Classifier Ensembles and Their Relationship with the Ensemble Accuracy. *Machine Learning*, 51(2):181–207, 2003.

[288] K. Lalit Narayan, K. Mallikarjuna Rao, and M. Sarcar. *Computer Aided Design and Manufacturing*. Prentice-Hall, 2008.

[289] A. Lamkanfi and S. Demeyer. Filtering Bug Reports for Fix-Time Analysis. In *Proc. of the 16th European Conference on Software Maintenance and Reengineering*, pages 379–384, 2012.

[290] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals. Predicting the Severity of a Reported Bug. In *Proc. of the 7th Working Conference on Mining Software Repositories*, pages 1–10, 2010.

[291] N. Lavesson and P. Davidsson. Quantifying the Impact of Learning Algorithm Parameter Tuning. In *Proc. of the 21st National Conference on Artificial Intelligence*, pages 395–400, 2006.

[292] S. Lehnert. A Review of Software Change Impact Analysis. Technical report, Ilmenau University of Technology, 2011.

[293] S. Lehnert. A Taxonomy for Software Change Impact Analysis. In *Proc. of the 12th International Workshop on Principles of Software Evolution and the 7th Annual ERCIM Workshop on Software Evolution*, pages 41–50, 2011.

[294] R. Lenth. Response-Surface Methods in R, Using rsm. *Journal of Statistical Software*, 32(7):1–17, 2009.

[295] T. Lethbridge, S. Sim, and J. Singer. Studying Software Engineers: Data Collection Techniques for Software Field Studies. *Empirical Software Engineering*, 10(3):311–341, 2005.

[296] D. Lettner, F. Angerer, H. Prähofer, and P. Grünbacher. A Case Study on Software Ecosystem Characteristics in Industrial Automation Software. In *Proc. of the 3rd International Conference on Software and System Process*, pages 40–49, 2014.

[297] J. Leuser. Challenges for Semi-Automatic Trace Recovery in the Automotive Domain. In *Proc. of the 7th International Workshop on Traceability in Emerging Forms of Software Engineering*, pages 31–35, 2009.

[298] J. Leuser and D. Ott. Tackling Semi-Automatic Trace Recovery for Large Specifications. In *Proc. of the 16th International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 203–217, 2010.

[299] S. Levy and D. Steinberg. Computer Experiments: A Review. *Advances in Statistical Analysis*, 94(4):311–324, 2010.

[300] D. Lewis. Naïve (Bayes) at Forty: The Independence Assumption in Information Retrieval. In *Machine Learning*, volume 1398, pages 4–15. 1998.

[301] 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.

[302] N. Li, Z. Li, Y. Nie, X. Sun, and X. Li. Predicting Software Black-Box Defects Using Stacked Generalization. In *Proc. of the 6th International Conference on Digital Information Management*, pages 294–299, 2011.

[303] Q. Li, Q. Wang, Y. Yang, and M. Li. Reducing Biases in Individual Software Effort Estimations: A Combining Approach. In *Proc. of the 2nd International Symposium on Empirical Software Engineering and Measurement*, pages 223–232, 2008.

[304] Y. Li, J. Li, Y. Yang, and M. Li. Requirement-Centric Traceability for Change Impact Analysis: A Case Study. In *Proc. of the 2nd International Conference on Software Process*, pages 100–111, 2008.

[305] Z. Li, M. Harman, and R. Hierons. Search Algorithms for Regression Test Case Prioritization. *Transactions on Software Engineering*, 33(4):225–237, 2007.

[306] E. Liddy. *Natural Language Processing*. Encyclopedia of Library and Information Science. Marcel Decker, 2nd edition, 2001.

[307] J. Lin, L. Chan, J. Cleland-Huang, R. Settimi, J. Amaya, G. Bedford, B. Berenbach, O. BenKhadra, D. Chuan, and X. Zou. Poirot: A Distributed Tool Supporting Enterprise-Wide Automated Traceability. In *Proc. of the 14th International Conference on Requirements Engineering*, pages 363–364, 2006.

[308] Z. Lin, F. Shu, Y. Yang, C. Hu, and Q. Wang. An Empirical Study on Bug Assignment Automation Using Chinese Bug Data. In *Proc. of the 3rd International Symposium on Empirical Software Engineering and Measurement*, pages 451–455, 2009.

[309] M. Linares-Vásquez, K. Hossen, H. Dang, H. Kagdi, M. Gethers, and D. Poshyvanyk. Triaging incoming change requests: Bug or commit history, or code authorship? In *Proc. of the 28th International Conference on Software Maintenance*, pages 451–460, 2012.

[310] M. Lindvall, R. Feldmann, G. Karabatis, Z. Chen, and V. Janeja. Searching for Relevant Software Change Artifacts Using Semantic Networks. In *Proc. of the 24th Symposium on Applied Computing*, pages 496–500, 2009.

[311] T. Liu. Learning to Rank for Information Retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009.

[312] T. Liu. *Learning to Rank for Information Retrieval*. Springer, 2011.

[313] S. Lohar, S. Amornborvornwong, A. Zisman, and J. Cleland-Huang. Improving Trace Accuracy Through Data-driven Configuration and Composition of Tracing Features. In *Proc. of the 9th Joint Meeting on Foundations of Software Engineering*, pages 378–388, 2013.

[314] M. Lormans and A. van Deursen. Reconstructing Requirements Coverage Views from Design and Test Using Traceability Recovery via LSI. In *Proc. of the 3rd International Workshop on Traceability in Emerging Forms of Software Engineering*, pages 37–42, 2005.

[315] M. Lormans and A. van Deursen. Can LSI Help Reconstructing Requirements Traceability in Design and Test? In *Proc. of the 10th European Conference on Software Maintenance and Reengineering*, pages 45–54, 2006.

[316] M. Lormans, A. van Deursen, and H. Gross. An Industrial Case Study in Reconstructing Requirements Views. *Empirical Software Engineering*, 13(6):727–760, 2008.

[317] M. Lubars, C. Potts, and C. Richter. A Review of the State of the Practice in Requirements Modeling. In *Proc. of the 1st International Symposium on Requirements Engineering*, pages 2–14, 1993.

[318] P. Lukacs, K. Burnham, and D. Anderson. Model Selection Bias and Freedman's Paradox. *Annals of the Institute of Statistical Mathematics*, 62(1):117–125, 2010.

[319] C. Macdonald, R. Santos, and I. Ounis. The Whens and Hows of Learning to Rank for Web Search. *Information Retrieval*, 16(5):584–628, 2013.

[320] R. Madachy. *Software Process Dynamics*. Wiley, 2007.

[321] A. Mahmoud and N. Niu. Using Semantics-Enabled Information Retrieval in Requirements Tracing: An Ongoing Experimental Investigation. In *Proc. of the 34th International Computer Software and Applications Conference*, pages 246–247, 2010.

[322] A. Mahmoud and N. Niu. Source Code Indexing for Automated Tracing. In *Proc. of the 6th International Workshop on Traceability in Emerging forms of Software Engineering*, pages 3–9, 2011.

[323] A. Maiga, A. Hamou-Lhadj, and A. Larsson. ReCRAC: A Recommender System for Crash Reports Assignment and Correction. In *Proc. of the 1st International Conference on Intelligent Systems, Data Mining and Information Technology*, pages 13–16, 2014.

[324] K. Malterud. The Art and Science of Clinical Knowledge: Evidence Beyond Measures and Numbers. *The Lancet*, 358(9279):397–400, 2001.

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

[326] A. Marcus and J. Maletic. Recovering Documentation-to-Source-Code Traceability Links Using Latent Semantic Indexing. In *Proc. of the 25th International Conference on Software Engineering*, pages 125–135, 2003.

[327] A. Marcus, J. Maletic, and A. Sergeyev. Recovery of Traceability Links Between Software Documentation and Source Code. *International Journal of Software Engineering and Knowledge Engineering*, 15(5):811–836, 2005.

[328] A. Marcus, A. Sergeyev, V. Rajlich, and J. Maletic. An Information Retrieval Approach to Concept Location in Source Code. In *Proc. of the 11th Working Conference on Reverse Engineering*, pages 214–223, 2004.

[329] M. Maron and J. Kuhns. On Relevance, Probabilistic Indexing and Information Retrieval. *Journal of the ACM*, 7(3):216–244, 1960.

[330] R. Martin and G. Melnik. Tests and Requirements, Requirements and Tests: A Möbius Strip. *IEEE Software*, 25(1):54–59, 2008.

[331] J. Matejka, E. Li, T. Grossman, and G. Fitzmaurice. CommunityCommands: Command Recommendations for Software Applications. In *Proc. of the 22nd Annual Symposium on User Interface Software and Technology*, pages 193–202, 2009.

[332] D. Matter, A. Kuhn, and O. Nierstrasz. Assigning Bug Reports Using a Vocabulary-Based Expertise Model of Developers. In *Proc. of the 6th International Working Conference on Mining Software Repositories*, pages 131–140, 2009.

[333] H. Mauritzon. *Automated Analysis of Large-scale Customer Support Issues*. MSc Thesis, to be submitted, Lund University, 2015.

[334] A. McCallum. MALLET: A Machine Learning for Language Toolkit. Technical report, 2002.

[335] M. McCandless, E. Hatcher, and O. Gospodnetic. *Lucene in Action*. Manning Publications, 2nd edition, 2010.

[336] G. Melnik, F. Maurer, and M. Chiasson. Executable Acceptance Tests for Communicating Business Requirements: Customer Perspective. In *Proc. of the 6th Agile Conference*, pages 12–46, 2006.

[337] T. Mens. Introduction and Roadmap: History and Challenges of Software Evolution. In T. Mens and S. Demeyer, editors, *Software Evolution*, pages 1–11. Springer, 2008.

[338] T. Mens, M. Wermelinger, S. Ducasse, S. Demeyer, R. Hirschfeld, and M. Jazayeri. Challenges in Software Evolution. In *Proc. of the 8th International Workshop on Principles of Software Evolution*, pages 13–22, 2005.

[339] T. Menzies and M. Shepperd. Special Issue on Repeatable Results in Software Engineering Prediction. *Empirical Software Engineering*, 17(1-2):1–17, 2012.

[340] A. Miller. *Subset Selection in Regression*. CRC Press, 2002.

[341] P. Mohagheghi and V. Dehlen. Where Is the Proof? - A Review of Experiences from Applying MDE in Industry. In *Proc. of the 4th European Conference on Model Driven Architecture: Foundations and Applications*, pages 432–443, 2008.

[342] D. Montgomery. *Design and Analysis of Experiments*. Wiley, 8th edition, 2013.

[343] K. Moon. The Nature of Computer Programs: Tangible? Goods? Personal Property? Intellectual Property? *European Intellectual Property Review*, 31(8):396–407, 2009.

[344] P. Morville. *Ambient Findability: What We Find Changes Who We Become*. O'Reilly Media, 2005.

[345] Mozilla Foundation. Bugzilla 5.1 Documentation, 2.4.4 Life Cycle of a Bug. Technical report.

[346] E. Murphy-Hill and G. Murphy. Recommendation Delivery. In M. Robillard, W. Maalej, R. Walker, and T. Zimmermann, editors, *Recommendation Systems in Software Engineering*, pages 223–242. Springer, 2014.

[347] R. Myers, D. Montgomery, and C. Anderson-Cook. *Response Surface Methodology: Process and Product Optimization Using Designed Experiments*. Wiley, 2009.

[348] T. Myklebust, T. Stålhane, G. Hanssen, and B. Haugset. Change Impact Analysis as Required by Safety Standards, What To Do? In *Proc. of the 12th Probabilistic Safety Assessment and Management Conference*, 2014.

[349] N. Nagwani and S. Verma. Predicting Expert Developers for Newly Reported Bugs Using Frequent Terms Similarities of Bug Attributes. In *Proc. of the 9th International Conference on ICT and Knowledge Engineering*, pages 113–117, 2012.

[350] S. Nair, J. de la Vara, M. Sabetzadeh, and L. Briand. An Extended Systematic Literature Review on Provision of Evidence for Safety Certification. *Information and Software Technology*, 56(7):689–717, 2014.

[351] M. Nandagopal, K. Gala, and V. Premnath. Improving Technology Commercialization at Research Institutes: Practical Insights from NCL Innovations. In *Proc. of the Innovation Educators' Conference*, 2011.

[352] J. Natt och Dag, V. Gervasi, S. Brinkkemper, and B. Regnell. Speeding Up Requirements Management in a Product Software Company: Linking Customer Wishes to Product Requirements through Linguistic Engineering. In *Proc. of the 12th International Requirements Engineering Conference*, pages 283–294, 2004.

[353] J. Natt och Dag, V. Gervasi, S. Brinkkemper, and B. Regnell. A Linguistic-Engineering Approach to Large-Scale Requirements Management. *IEEE Software*, 22(1):32–39, 2005.

[354] J. Natt och Dag, B. Regnell, P. Carlshamre, M. Andersson, and J. Karlsson. A Feasibility Study of Automated Natural Language Requirements Analysis in Market-Driven Development. *Requirements Engineering*, 7(1):20–33, 2002.

[355] J. Natt och Dag, T. Thelin, and B. Regnell. An Experiment on Linguistic Tool Support for Consolidation of Requirements from Multiple Sources in Market-Driven Product Development. *Empirical Software Engineering*, 11(2):303–329, 2006.

[356] C. Nebut, F. Fleurey, Y. Le Traon, and J. Jezequel. Automatic Test Generation: A Use Case Driven Approach. *Transactions on Software Engineering*, 32(3):140–155, 2006.

[357] G. Neddermeijer, G. van Oortmarssen, N. Piersma, and R. Dekker. A Framework for Response Surface Methodology for Simulation Optimization. In *Proc. of the 32nd Conference on Winter Simulation*, pages 129–136, 2000.

[358] J. Nievergelt. Exhaustive Search, Combinatorial Optimization and Enumeration: Exploring the Potential of Raw Computing Power. In *Proc. of the 27th Conference on Current Trends in Theory and Practice of Informatics*, pages 18–35, 2000.

[359] R. Oliveto. *Traceability Management Meets Information Retrieval Methods: Strengths and Limitations*. PhD Thesis, University of Salerno, 2008.

[360] R. Oliveto, M. Gethers, D. Poshyvanyk, and A. De Lucia. On the Equivalence of Information Retrieval Methods for Automated Traceability Link Recovery. In *Proc. of the 18th International Conference on Program Comprehension*, pages 68–71, 2010.

[361] L. Olofsson and P. Gullin. *Development of a Decision Support System for Defect Reports*. MSc Thesis, Lund University, http://sam.cs.lth.se/ExjobGetFile?id=658, 2014.

[362] T. Olsson. *Software Information Management in Requirements and Test Documentation*. Licentiate Thesis, Lund University, 2002.

[363] A. Orso, T. Apiwattanapong, and M. Harrold. Leveraging Field Data for Impact Analysis and Regression Testing. In *Proc. of the 9th European Software Engineering Conference*, pages 128–137, 2003.

[364] S. Owen, R. Anil, T. Dunning, and E. Friedman. *Mahout in Action*. Manning Publications, 2011.

[365] F. Paci, F. Massacci, F. Bouquet, and S. Debricon. Managing Evolution by Orchestrating Requirements and Testing Engineering Processes. In *Proc. of the 5th International Conference on Software Testing, Verification and Validation*, pages 834–841, 2012.

[366] S. Panichella. *Supporting Newcomers in Software Development Projects*. PhD thesis, University of Sannio, 2014.

[367] R. Parasuraman, T. Sheridan, and C. Wickens. A Model for Types and Levels of Human Interaction with Automation. *Transactions on Systems, Man and Cybernetics*, 30(3):286–297, 2000.

[368] J. Park, M. Lee, J. Kim, S. Hwang, and S. Kim. COSTRIAGE: A Cost-Aware Triage Algorithm for Bug Reporting Systems. In *Proc. of the 25th AAAI Conference on Artificial Intelligence*, 2011.

[369] S. Park, H. Kim, Y. Ko, and J. Seo. Implementation of an Efficient Requirements Analysis Supporting System Using Similarity Measure Techniques. *Information and Software Technology*, 42(6):429–438, 2000.

[370] A. Parvathy, B. Vasudevan, and R. Balakrishnan. A Comparative Study of Document Correlation Techniques for Traceability Analysis. In *Proc. of the 10th International Conference on Enterprise Information Systems, Information Systems Analysis and Specification*, pages 64–69, 2008.

[371] J. Paulson, G. Succi, and A. Eberlein. An Empirical Study of Open-Source and Closed-Source Software Products. *Transactions on Software Engineering*, 30(4):246–256, 2004.

[372] D. Perry, A. Porter, and L. Votta. Empirical Studies of Software Engineering: A Roadmap. In *Proc. of 22nd International Conference on Software Engineering*, pages 345–355, 2000.

[373] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson. Systematic Mapping Studies in Software Engineering. In *Proc. of the 12th International Conference on Evaluation and Assessment in Software Engineering*, pages 71–80, 2008.

[374] K. Petersen and C. Wohlin. Context in Industrial Software Engineering Research. In *Proc. of the 3rd International Symposium on Empirical Software Engineering and Measurement*, pages 401–404, 2009.

[375] F. Pettersson, M. Ivarsson, T. Gorschek, and P. Öhman. A Practitioner's Guide to Light Weight Software Process Assessment and Improvement Planning. *Journal of Systems and Software*, 81(6):972–995, 2008.

[376] S. Pfleeger. Experimental Design and Analysis in Software Engineering. *Annals of Software Engineering*, 1(1):219–253, 1995.

[377] K. Pohl, G. Böckle, and F. van der Linden. *Software Product Line Engineering: Foundations, Principles, and Techniques*. Birkhäuser, 2005.

[378] J. Ponte and B. Croft. A Language Modeling Approach to Information Retrieval. In *Proc. of the 21st Annual International SIGIR Conference on Research and Development in Information Retrieval*, pages 275–281, 1998.

[379] H. Post, C. Sinz, F. Merz, T. Gorges, and T. Kropf. Linking Functional Requirements and Software Verification. In *Proc. of the 17th Requirements Engineering Conference*, pages 295–302, 2009.

[380] D. Power. Understanding Data-Driven Decision Support Systems. *Information Systems Management*, 25(2):149–154, 2008.

[381] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, 2008.

[382] Radio Technical Commission for Aeronautics. DO-178C Software Considerations in Airborne Systems and Equipment Certification. Technical report, 2012.

[383] B. Ramesh. Factors Influencing Requirements Traceability Practice. *Communications of the ACM*, 41(12):37–44, 1998.

[384] B. Ramesh and M. Jarke. Toward Reference Models for Requirements Traceability. *Transactions on Software Engineering*, 27(1):58–93, 2001.

[385] B. Ramesh, C. Stubbs, T. Powers, and M. Edwards. Requirements Traceability: Theory and Practice. *Annals of Software Engineering*, 3(1):397–415, 1997.

[386] B. Randall. Towards a Methodology of Computing System Design. In P. Naur and B. Randall, editors, *NATO Working Conference on Software Engineering 1968, Report on a Conference Sponsored by NATO Scientific Committee*, pages 204–208. 1969.

[387] J. Randolph. Free-Marginal Multirater Kappa: An Alternative to Fleiss' Fixed-Marginal Multirater Kappa. In *Joensuu Learning and Instruction Symposium*, 2005.

[388] R. Rao, G. Fung, and R. Rosales. On the Dangers of Cross-Validation. An Experimental Evaluation. In *Proc. of the 8th SIAM International Conference on Data Mining*, pages 588–596, 2008.

[389] S. Rao and A. Kak. Retrieval from Software Libraries for Bug Localization: A Comparative Study of Generic and Composite Text Models. In *Proc. of the 8th Working Conference on Mining Software Repositories*, pages 43–52, 2011.

[390] B. Regnell, R. Berntsson Svensson, and T. Olsson. Supporting Roadmapping of Quality Requirements. *IEEE Software*, 25(2):42–47, 2008.

[391] B. Regnell, R. Berntsson Svensson, and K. Wnuk. Can We Beat the Complexity of Very Large-Scale Requirements Engineering? In *Proc. of the 14th International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 123–128, 2008.

[392] B. Regnell and P. Runeson. Combining Scenario-based Requirements with Static Verification and Dynamic Testing. In *Proc. of the 4th International Workshop on Requirements Engineering: Foundation for Software Quality*, pages 195–206, 1998.

[393] B. Regnell, P. Runeson, and C. Wohlin. Towards Integration of Use Case Modelling and Usage-based Testing. *Journal of Systems and Software*, 50(2):117–130, 2000.

[394] P. Rempel, P. Mäder, T. Kuschke, and J. Cleland-Huang. Mind the Gap: Assessing the Conformance of Software Traceability to Relevant Guidelines. In *Proc. of the 36th International Conference on Software Engineering*, pages 943–954, 2014.

[395] A. Rencher. *Methods of Multivariate Analysis*. Wiley, 2002.

[396] F. Ricca, M. Torchiano, M. Di Penta, M. Ceccato, and P. Tonella. Using Acceptance Tests as a Support for Clarifying Requirements: A Series of Experiments. *Information and Software Technology*, 51(2):270–283, 2009.

[397] S. Robertson. The Probability Ranking Principle in IR. *Journal of Documentation*, 33(4):294–304, 1977.

[398] S. Robertson and S. Jones. Relevance Weighting of Search Terms. *Journal of the American Society for Information Science*, 27(3):129–146, 1976.

[399] S. Robertson and J. Robertson. *Mastering the Requirements Process*. Addison-Wesley Professional, 1999.

[400] S. Robertson and H. Zaragoza. The Probabilistic Relevance Framework: BM25 and Beyond. *Foundation and Trends in Information Retrieval*, 3(4):333–389, 2009.

[401] M. Robillard, W. Maalej, R. Walker, and T. Zimmermann, editors. *Recommendation Systems in Software Engineering*. Springer, 2014.

[402] M. Robillard and R. Walker. An Introduction to Recommendation Systems in Software Engineering. In M. Robillard, W. Maalej, R. Walker, and T. Zimmermann, editors, *Recommendation Systems in Software Engineering*, pages 1–11. Springer, 2014.

[403] M. Robillard, R. Walker, and T. Zimmermann. Recommendation Systems for Software Engineering. *IEEE Software*, 27(4):80–86, 2010.

[404] B. Robinson and P. Francis. Improving Industrial Adoption of Software Engineering Research: A Comparison of Open and Closed Source Software. In *Proc. of the International Symposium on Empirical Software Engineering and Measurement*, volume 21, pages 1–10, 2010.

[405] H. Robinson, J. Segal, and H. Sharp. Ethnographically-Informed Empirical Studies of Software Practice. *Information and Software Technology*, 49(6):540–551, 2007.

[406] G. Robles and J. González-Barahona. Contributor Turnover in Libre Software Projects. In E. Damiani, B. Fitzgerald, W. Scacchi, M. Scotto, and G. Succi, editors, *Open Source Systems*, pages 273–286. Springer, 2006.

[407] B. Robson. *Real World Research*. Blackwell, 2nd edition, 2002.

[408] J. Rocchio. Relevance Feedback in Information Retrieval. In G. Salton, editor, *The SMART Retrieval System: Experiments in Automatic Document Processing*, pages 313–323. Prentice-Hall, 1971.

[409] P. Rovegård, L. Angelis, and C. Wohlin. An Empirical Study on Views of Importance of Change Impact Analysis Issues. *Transactions on Software Engineering*, 34(4):516–530, 2008.

[410] P. Runeson, M. Alexandersson, and O. Nyholm. Detection of Duplicate Defect Reports Using Natural Language Processing. In *Proc. of the 29th International Conference on Software Engineering*, pages 499–510, 2007.

[411] P. Runeson, C. Andersson, and M. Höst. Test Processes in Software Product Evolution: A Qualitative Survey on the State of Practice. *Journal of Software Maintenance*, 15(1):41–59, 2003.

[412] P. Runeson and M. Höst. Guidelines for Conducting and Reporting Case Study Research in Software Engineering. *Empirical Software Engineering*, 14(2):131–164, 2009.

[413] P. Runeson, M. Höst, A. Rainer, and B. Regnell. *Case Study Research in Software Engineering. Guidelines and Examples*. Wiley, 2012.

[414] G. Sabaliauskaite, A. Loconsole, E. Engström, M. Unterkalmsteiner, B. Regnell, P. Runeson, T. Gorschek, and R. Feldt. Challenges in Aligning Requirements Engineering and Verification in a Large-Scale Industrial Context. In *Proc. of the 16th International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 128–142, 2010.

[415] A. Said, D. Tikk, and P. Cremonesi. Benchmarking. In M. Robillard, W. Maalej, R. Walker, and T. Zimmermann, editors, *Recommendation Systems in Software Engineering*, pages 275–300. Springer, 2014.

[416] G. Salton and C. Buckley. Term-Weighting Approaches in Automatic Text Retrieval. *Information Processing and Management*, 24(5):513–523, 1988.

[417] G. Salton, A. Wong, and C. Yang. A Vector Space Model for Automatic Indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[418] T. Santner, B. Williams, and W. Notz. *The Design and Analysis of Computer Experiments*. Springer, 2003.

[419] W. Scacchi. Understanding the Requirements for Developing Open Source Software Systems. *IEEE Software*, 149(1):24–39, 2002.

[420] B. Schaffer and C. Riordan. A Review of Cross-Cultural Methodologies for Organizational Research: A Best-Practices Approach. *Organizational Research Methods*, 6(2):169–215, 2003.

[421] D. Sculley. Large Scale Learning to Rank. In *Proc. of the Workshop on Advances in Ranking at the 23rd NIPS Conference*, pages 1–6, 2009.

[422] C. Seaman. Qualitative Methods in Empirical Studies of Software Engineering. *Transactions on Software Engineering*, 25(4):557–572, 1999.

[423] C. Seiffert, T. Khoshgoftaar, J. Van Hulse, and A. Folleco. An Empirical Study of the Classification Performance of Learners on Imbalanced and Noisy Software Quality Data. *Information Sciences*, 259:571–595, 2014.

[424] F. Servant and J. Jones. WhoseFault: Automatic Developer-to-Fault Assignment Through Fault Localization. In *Proc. of the 34th International Conference on Software Engineering*, pages 36–46, 2012.

[425] R. Settimi, J. Cleland-Huang, O. BenKhadra, J. Mody, W. Lukasik, and C. De Palma. Supporting Software Evolution Through Dynamically Retrieving Traces to UML Artifacts. In *Proc. of the 7th International Workhop on Principles of Software Evolution*, pages 49–54, 2004.

[426] M. Shepperd, C. Schofield, and B. Kitchenham. Effort Estimation Using Analogy. In *Proc. of the 18th International Conference on Software Engineering*, pages 170–178, 1996.

[427] M. Shepperd, Q. Song, Z. Sun, and C. Mair. Data Quality: Some Comments on the NASA Software Defect Datasets. *Transactions on Software Engineering*, 39(9):1208–1215, 2013.

[428] R. Shokripour, Z. Kasirun, S. Zamani, and J. Anvik. Automatic Bug Assignment Using Information Extraction Methods. In *Proc. of the 1st International Conference on Advanced Computer Science Applications and Technologies*, pages 144–149, 2012.

[429] F. Shull, J. Carver, S. Vegas, and N. Juristo. The Role of Replications in Empirical Software Engineering. *Empirical Software Engineering*, 13(2):211–218, 2008.

[430] F. Shull, J. Singer, and D. Sjøberg. *Guide to Advanced Empirical Software Engineering*. Springer, 2010.

[431] E. Sikora, B. Tenbergen, and K. Pohl. Industry Needs and Research Directions in Requirements Engineering for Embedded Systems. *Requirements Engineering*, 17(1):57–78, 2012.

[432] J. Sill, G. Takács, L. Mackey, and D. Lin. Feature-Weighted Linear Stacking. *http://arxiv.org/abs/0911.0460*, 2009.

[433] A. Singhal. Modern Information Retrieval: A Brief Overview. *Data Engineering Bulletin*, 24(2):1–9, 2001.

[434] A. Smeaton and D. Harman. The TREC Experiments and Their Impact on Europe. *Journal of Information Science*, 23(2):169–174, 1997.

[435] J. Sowa. Semantic Networks. In *Encyclopedia of Cognitive Science*. Wiley, 2006.

[436] G. Spanoudakis, A. d'Avila Garcez, and A. Zisman. Revising Rules to Capture Requirements Traceability Relations: A Machine Learning Approach. In *Proc. of the 15th International Conference in Software Engineering and Knowledge Engineering*, pages 570–577, 2003.

[437] G. Spanoudakis, A. Zisman, E. Pérez-Miñana, and P. Krause. Rule-Based Generation of Requirements Traceability Relations. *Journal of Systems and Software*, 72(2):105–127, 2004.

[438] K. Spärck Jones, S. Walker, and S. Robertson. A Probabilistic Model of Information Retrieval: Development and Comparative Experiments. *Information Processing and Management*, 36(6):779–808, 2000.

[439] A. Stone and P. Sawyer. Using Pre-Requirements Tracing to Investigate Requirements Based on Tacit Knowledge. In *Proc. of the 1st International Conference on Software and Data Technologies*, pages 139–144, 2006.

[440] T. Stålhane, G. Hanssen, T. Myklebust, and B. Haugset. Agile Change Impact Analysis of Safety Critical Software. In *Proc. of the International Workshop on Next Generation of System Assurance Approaches for Safety-Critical Systems*, pages 444–454, 2014.

[441] S. Sulaman Muhammad, A. Oručević-Alagic, M. Borg, K. Wnuk, M. Höst, and J. de la Vara. Development of Safety-Critical Software Systems Using Open Source Software - A Systematic Map. In *Proc. of the 40th Euromicro Conference on Software Engineering and Advanced Applications*, pages 17–24, 2014.

[442] H. Sultanov and J. Huffman Hayes. Application of Swarm Techniques to Requirements Engineering: Requirements Tracing. In *Proc. of the 18th International Requirements Engineering Conference*, pages 211–220, 2010.

[443] S. Sundaram, J. Huffman Hayes, A. Dekhtyar, and A. Holbrook. Assessing Traceability of Software Engineering Artifacts. *Requirements Engineering*, 15(3):313–335, 2010.

[444] A. Tamrawi, T. Nguyen, J. Al-Kofahi, and T. Nguyen. Fuzzy Set and Cache-Based Approach for Bug Triaging. In *Proc. of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, pages 365–375, 2011.

[445] P. Teetor. *R Cookbook*. O'Reilly Media, 2011.

[446] S. Thomas, M. Nagappan, D. Blostein, and A. Hassan. The Impact of Classifier Configuration and Classifier Combination on Bug Localization. *Transactions on Software Engineering*, 39(10):1427–1443, 2013.

[447] R. Tibshirani, G. Walther, and T. Hastie. Estimating the Number of Clusters in a Data Set via the Gap Statistic. *Journal of the Royal Statistical Society*, 63(2):411–423, 2001.

[448] M. Torchiano and F. Ricca. Impact Analysis by Means of Unstructured Knowledge in the Context of Bug Repositories. In *Proc. of the 4th International Symposium on Empirical Software Engineering and Measurement*, pages 1–4, 2010.

[449] R. Torkar, T. Gorschek, R. Feldt, M. Svahnberg, U. Raja, and K. Kamran. Requirements Traceability: A Systematic Review and Industry Case Study. *International Journal of Software Engineering and Knowledge Engineering*, 22(3):1–49, 2012.

[450] A. Tosun Misirli, A. Bener, B. Caglayan, G. Calikli, and B. Turhan. Field Studies - A Methodology for Construction and Evaluation of Recommendation Systems in Software Engineering. In M. Robillard, W. Maalej, R. Walker, and T. Zimmermann, editors, *Recommendation Systems in Software Engineering*, pages 329–355. Springer, 2014.

[451] G. Travassos and M. Barros. Contributions of In Virtuo and In Silico Experiments for the Future of Empirical Studies in Software Engineering. In

Proc. of the 2nd Workshop on Empirical Studies in Software Engineering, pages 117–130, 2003.

[452] G. Travassos, P. dos Santos, P. Neto, and J. Biolchini. An Environment to Support Large Scale Experimentation in Software Engineering. In *Proc. of the 13th International Conference on Engineering of Complex Computer Systems*, pages 193–202, 2008.

[453] M. Tsunoda and K. Ono. Pitfalls of Analyzing a Cross-Company Dataset of Software Maintenance and Support. In *Proc. of the 15th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, pages 1–6, 2014.

[454] B. Turhan. On the Dataset Shift Problem in Software Engineering Prediction Models. *Empirical Software Engineering*, 17(1-2):62–74, 2012.

[455] H. Turtle and B. Croft. Evaluation of an Inference Network-Based Retrieval Model. *Transactions on Information Systems*, 9(3):187–222, 1991.

[456] M. Unterkalmsteiner, R. Feldt, and T. Gorschek. A Taxonomy for Requirements Engineering and Software Test Alignment. *Transactions on Software Engineering Methodology*, 23(2):16:1–38, 2014.

[457] J. Urbano. Information Retrieval Meta-Evaluation: Challenges and Opportunities in the Music Domain. In *Proc. of the 12th International Society for Music Information Retrieval Conference*, pages 597–602, 2011.

[458] E. Uusitalo, M. Komssi, M. Kauppinen, and A. Davis. Linking Requirements and Testing in Practice. In *Proc. of the 16th International Requirements Engineering Conference*, pages 265–270, 2008.

[459] S. Vaidhyanathan. *The Googlization of Everything: (And Why We Should Worry)*. University of California Press, 2012.

[460] C.J. van Rijsbergen. *Information Retrieval*. Butterworth, 1979.

[461] B. Van Rompaey and S. Demeyer. Establishing Traceability Links between Unit Test Cases and Units under Test. In *Proc. of the 13th European Conference on Software Maintenance and Reengineering*, pages 209–218, 2009.

[462] M. Vierhauser, R. Rabiser, and P. Grünbacher. A Case Study on Testing, Commissioning, and Operation of Very-Large-Scale Software Systems. In *Proc. of the 36th International Conference on Software Engineering*, pages 125–134, 2014.

[463] G. Vining. Adapting Response Surface Methodology for Computer and Simulation Experiments. In H. Tsubaki, S. Yamada, and K. Nishina, editors, *The Grammar of Technology Development*, pages 127–134. Springer, 2008.

[464] A. von Knethen and M. Grund. QuaTrace: A Tool Environment for (Semi)-Automatic Impact Analysis Based on Traces. In *Proc. of the 19th International Conference on Software Maintenance*, pages 246–255, 2003.

[465] E. Voorhees. *TREC: Experiment and Evaluation in Information Retrieval*. MIT Press, 2005.

[466] R. Walker and R. Holmes. Simulation - A Methodology to Evaluate Recommendation Systems in Software Engineering. In M. Robillard, W. Maalej, R. Walker, and T. Zimmermann, editors, *Recommendation Systems in Software Engineering*, pages 301–327. Springer, 2014.

[467] X. Wang, G. Lai, and C. Liu. Recovering Relationships between Documentation and Source Code based on the Characteristics of Software Engineering. *Electronic Notes in Theoretical Computer Science*, 243:121–137, 2009.

[468] R. Watkins and M. Neal. Why and How of Requirements Tracing. *IEEE Software*, 11(4):104–106, 1994.

[469] T. Wetzlmaier and R. Ramler. Improving Manual Change Impact Analysis with Tool Support: A Study in an Industrial Project. In *Proc. of the 7th Software Quality Days*, pages 47–66, 2015.

[470] H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer, 2009.

[471] R. Wieringa. *Design Science Methodology for Information Systems and Software Engineering*. Springer, 2014.

[472] K. Wiklund, S. Eldh, D. Sundmark, and K. Lundqvist. Can We Do Useful Industrial Software Engineering Research in the Shadow of Lean and Agile? In *Proc. of the 1st International Workshop on Conducting Empirical Studies in Industry*, pages 67–68, 2013.

[473] S. Winkler and J. Pilgrim. A Survey of Traceability in Requirements Engineering and Model-Driven Development. *Software & Systems Modeling*, 9(4):529–565, 2010.

[474] I. Witten, E. Frank, and M. Hall. *Data Mining*. Morgan Kaufmann, 2011.

[475] K. Wnuk, T. Gorschek, and S. Zahda. Obsolete Software Requirements. *Information and Software Technology*, 55(6):921–940, 2013.

[476] C. Wohlin. Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering. In *Proc. of the 18th International Conference on Evaluation and Assessment in Software Engineering*, pages 1–10, 2014.

[477] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering: A Practical Guide*. Springer, 2012.

[478] D. Wolpert. Stacked Generalization. *Neural Networks*, 5(2):241–259, 1992.

[479] D. Wolpert and W. Macready. No Free Lunch Theorems for Optimization. *Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[480] W. Wong, V. Debroy, A. Surampudi, H. Kim, and M. Siok. Recent Catastrophic Accidents: Investigating How Software was Responsible. In *Proc. of the 4th International Conference on Secure Software Integration and Reliability Improvement*, pages 14–22, 2010.

[481] W. Wu, W. Zhang, Y. Yang, and Q. Wang. DREX: Developer Recommendation with K-Nearest-Neighbor Search and Expertise Ranking. In *Proc. of the 18th Asia Pacific Software Engineering Conference*, pages 389–396, 2011.

[482] X. Xia, D. Lo, X. Wang, and B. Zhou. Accurate Developer Recommendation for Bug Resolution. In *Proc. of the 20th Working Conference on Reverse Engineering*, pages 72–81, 2013.

[483] J. Xiao, R. Catrambone, and J. Stasko. Be Quiet? Evaluating Proactive and Reactive User Interface Assistants. In *Proc. of the 10th International Conference on Human-Computer Interaction*, pages 383–390, 2003.

[484] X. Xie, W. Zhang, Y. Yang, and Q. Wang. DRETOM: Developer Recommendation Based on Topic Models for Bug Resolution. In *Proc. of the 8th International Conference on Predictive Models in Software Engineering*, pages 19–28, 2012.

[485] R. Yin. *Case Study Research: Design and Methods*. Sage Publications, 3rd edition, 2003.

[486] A. Ying, G. Murphy, R. Ng, and M. Chu-Carroll. Predicting Source Code Changes by Mining Change History. *Transactions on Software Engineering*, 30(9):574–586, 2004.

[487] T. Yue, L. Briand, and Y. Labiche. A Systematic Review of Transformation Approaches between User Requirements and Analysis Models. *Requirements Engineering*, 16(2):75–99, 2011.

[488] M. Zaharia, N. Mosharaf Chowdhury, M. Franklin, S. Shenker, and I. Stoica. Spark: Cluster Computing with Working Sets. Technical report, EECS Department, University of California, Berkeley, California, 2010.

[489] H. Zaragoza and M. Najork. Web Search Relevance Ranking. In L. Liu and T. Oszu, editors, *Encyclopedia of Database Systems*, pages 3497–3501. Springer, 2009.

[490] C. Zhai. A Brief Review of Information Retrieval Models. Technical Report, University of Illinois at Urbana-Champaign, 2007.

[491] C. Zhai. Statistical Language Models for Information Retrieval A Critical Review. *Foundations and Trends in Information Retrieval*, 2(3):137–213, 2008.

[492] C. Zhai and J. Lafferty. Model-Based Feedback in the Language Modeling Approach to Information Retrieval. In *Proc. of the 10th International Conference on Information and Knowledge Management*, pages 403–410, 2001.

[493] J. Zhang, X. Wang, D. Hao, B. Xie, L. Zhang, and H. Mei. A Survey on Bug-report Analysis. *Science China Information Sciences*, 58(2):1–24, 2015.

[494] W. Zhao, L. Zhang, Y. Liu, J. Luo, and J. Sun. Understanding How the Requirements are Implemented in Source Code. In *Proc. of the 10th Asia-Pacific Software Engineering Conference*, pages 68–77, 2003.

[495] Y. Zhao and Y. Zhang. Comparison of Decision Tree Methods for Finding Active Objects. *Advances in Space Research*, 41(12):1955–1959, 2008.

[496] X. Zhou and H. Yu. A Clustering-Based Approach for Tracing Object-Oriented Design to Requirement. In *Proc. of the 10th International Conference on Fundamental Approaches to Software Engineering*, pages 412–422, 2007.

[497] T. Zimmermann, P. Weibgerber, S. Diehl, and A. Zeller. Mining Version Histories to Guide Software Changes. In *Proc. of the 26th International Conference on Software Engineering*, pages 563 – 572, 2004.

[498] X. Zou, R. Settimi, and J. Cleland-Huang. Phrasing in Dynamic Requirements Trace Retrieval. In *Proc. of the 30th International Computer Software and Applications Conference*, pages 265–272, 2006.

[499] X. Zou, R. Settimi, and J. Cleland-Huang. Evaluating the Use of Project Glossaries in Automated Trace Retrieval. In *Proc. of the International Conference on Software Engineering Research and Practice*, pages 157–163, 2008.

[500] X. Zou, R. Settimi, and J. Cleland-Huang. Improving Automated Requirements Trace Retrieval: A Study of Term-Based Enhancement Methods. *Empirical Software Engineering*, 15(2):119–146, 2010.