Supporting Change Impact Analysis Using a Recommendation System: An Industrial Case Study in a Safety-Critical Context

Markus Borg, *Member, IEEE,* Krzysztof Wnuk, Björn Regnell, and Per Runeson, *Member, IEEE*

Abstract—Change Impact Analysis (CIA) during software evolution of safety-critical systems is a labor-intensive task. Several authors have proposed tool support for CIA, but very few tools were evaluated in industry. We present a case study on ImpRec, a recommendation System for Software Engineering (RSSE), tailored for CIA at a process automation company. ImpRec builds on assisted tracing, using information retrieval solutions and mining software repositories to recommend development artifacts, potentially impacted when resolving incoming issue reports. In contrast to the majority of tools for automated CIA, ImpRec explicitly targets development artifacts that are not source code. We evaluate ImpRec in a two-phase study. First, we measure the correctness of ImpRec's recommendations by a simulation based on 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 on different continents. The 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.

Index Terms—CASE, maintenance management, software and system safety, tracing.

1 Introduction

Large-scale software-intensive systems evolve for years, if not decades. Several studies report that software evolution might last for over a decade, e.g., studies conducted at Siemens [1], Ericsson [2], and ABB [3]. Long-term evolution results in complex legacy systems in which changes are known to be labor-intensive [4] and error-prone [5]. Furthermore, inadequate change management is reported as one of the root causes for recent catastrophic failures caused by software [6]. Consequently, understanding how changes propagate in large systems is fundamental in software evolution.

In safety-critical software engineering, formal Change Impact Analysis (CIA) is mandated in process standards such as the general standard IEC 61508 [7], IEC 61511 in the process industry sector [8], ISO 26262 in the automotive domain [9], DO-178C in avionics [10], and EN 50128 for the railway sector [11]. 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 CIA implementation varies between development organizations, but remains an important component of the *safety case* [12], i.e., the structured argument evaluated by external assessors to justify safety certification [13].

Several studies report that CIA is a tedious part of software maintenance and evolution, e.g., in process automation [14], in nuclear power [15], in the automotive industry [16], and in the aerospace domain [17]. Still, the level of CIA tool support in industry is low, as shown in a recent cross-domain survey [18]. The low level of CIA automation has also been reported in previous work by Lettner *et al.* [19]. Moreover, Li *et al.* report that most CIA tools are restricted to source code impact, although other artifacts are also important, e.g., UML models [20], quality requirements [21], and regression test plans [22]. This paper contributes to filling this gap by explicitly focusing on *CIA of software artifacts that are not source code*, i.e., non-code artifacts.

Semi-automated tracing based on Information Retrieval (IR) techniques has the potential to support CIA. A handful of controlled experiments with student subjects show favorable results [23], [24], [25], and initial evidence from a small industrial case study suggests that IR-based tracing tools have the potential to support CIA [26]. However, still no tool implementing IR-based tracing has been evaluated in a large software development context [27].

Although scalability was identified as one of the eight "*grand challenges of traceability*" by CoEST[1], most previous evaluations on IR-based tracing studied small datasets containing fewer than 500 artifacts [27]. Instead, scalability is often discussed as a threat to validity, and highlighted as an important concern for future work (see, e.g., De Lucia *et al.* [28] and Huffman Hayes *et al.* [29]). The work presented in this paper focuses on investigating the scalability to industrial size contexts. 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

- *M. Borg is with SICS Swedish ICT AB, Ideon Science Park, Building Beta 2, Scheelevägen 17, SE-223 70 Lund, Sweden. This research was carried out while the author was with Lund University.*
  E-mail: markus.borg@sics.se
- *K. Wnuk is with Blekinge Institute of Technology, Sweden. B. Regnell and P. Runeson are with Lund University, Sweden.*

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).

in a real world context.

We have previously proposed an IR-based tracing tool enhanced by a semantic network mined from historical CIA data recorded in an issue tracker. Our combined approach is implemented in *ImpRec* [30], a Recommendation System for Software Engineering (RSSE) [31]. ImpRec is tailored for one of our industry partners active in process automation, but still comprises general solution elements. The RSSE assists developers to perform 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 centered around these issue reports.

In this paper, 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*" [31, pp. 9]. Our study responds to their call for comprehensive evaluations, by reporting from a longitudinal *in situ* study of ImpRec deployed in two proprietary development teams. Also, we perform a static validation [32] of ImpRec, resulting in quantitative results reflecting common evaluation practice in both RSSE and traceability research. We conclude that ImpRec can support CIA in industry, and that the tool implementation scales to the industrial context under study.

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).
- An in-depth case study, comprising both a quantitative static validation and a qualitative dynamic validation (Section 5). Thus, our study is designed to assess both the *correctness* of the ImpRec output, and the *utility* of actually working with ImpRec.
- Study results that show that ImpRec recommends about 40% of the true impact among the top-10 recommendations (Section 6.1). Also, that developers confirm 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 most similar 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 their counterparts in other engineering disciplines. Unfortunately, understanding the impact of changes to complex software systems is tedious, typically avoided unless necessary [4], and instead mended by extensive regression testing [33]. Thus, CIA is mandated in safety-critical software development practice [7], [9], [11] to avoid unpredictable consequences. Bohner defines CIA as "*identifying the potential consequences of a change, or estimating what needs to be modified to accomplish a change*" [34, pp. 3]. 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, Bohner 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. Our 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 [18]. 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. Improving CIA tools has also been highlighted by other researchers, 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*" [4, pp. 268], and Lehnert: "*more attention should be paid on linking requirements, architectures, and code to enable comprehensive CIA*" [35, pp. 26].

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 [36], partly based on Arnold and Bohner's early work on a framework for CIA comparison [37], and populated this taxonomy with 150 primary studies from a literature review [35]. The results show that only 13% of the studies address multiple artifact types, and 65% targets only source code.

The approaches for supporting CIA on the source code can be divided into static and dynamic. Examples of static approaches are to use a Global Hierarchical Object Graph to mine and rank dependencies to be analyzed during CIA [38] or to consider variable granularity of classes, the granularity of class members, and the granularity of code fragments for improving precision of CIA [39]. Examples of dynamic CIA approaches include dynamically examining program paths, yet limiting impact to observed call orders and call-return sequences compressed using the SEQUITUR data compression algorithm [40] or analyzing the coverage impact [41]. These approaches show promising results in supporting CIA and reducing tedious manual work, but are limited to source code and evaluated or relatively small cases compared to our case.

Several researchers focused on supporting CIA for requirements expressed as models. For example, operational dependencies between use cases and goals were used to propagate change between behavioral requirements and intentional requirements [42]. Briand *et al.* proposed a method to automatically identify changes between two different versions of UML models, according to a change taxonomy [20]. Cleland-Huang *et al.* focused on supporting CIA between functional and non-functional requirements with the help of soft goal dependencies [43]. The approaches are evaluated in case studies with smaller empirical contexts than the one presented in this paper.

Li *et al.* also concluded that the source code focus dominates among CIA researchers [44]. 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 industrial software maintenance context. Moreover, Li *et al.* explicitly suggested studies on tool support that provides ranked lists of potentially impacted entities, like the study on ImpRec we present in this paper. Gethers *et al.* presented another overview of tool support for CIA on the source code level [45].

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 [3]. Imp was evaluated on a large proprietary software system, and this is the first reported large-scale evaluation of tool support for 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. Ghosh *et al.* presented another study on CIA for software-intensive systems in the process automation domain [46]. The authors studied 33 historical software development projects to explore CIA triggered by requirement changes. Their focus is however not on what specific artifacts are impacted, but how much effort the changes require. Using a linear regression model, they predict the cost of changing requirements, and practitioners confirm the value of their approach.

Other research focuses on processes, particularly agile approaches. Stålhane *et al.* suggest supporting CIA by *process improvements*, and propose agile CIA as part of SafeScrum [47]. Their work originates from 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 [48]. Jonsson *et al.* also studied how to apply agile software development in the railway domain and conclude that CIA can be problematic in highly iterative development contexts [49].

Some researchers studied organizational aspects of CIA. Kilpinen identified that experiential CIA is common in industry, i.e., scoping changes through inspection and engineering judgement [50]. She introduced "*the adapted rework cycle*" to improve the interplay between CIA and the design process, and demonstrated its feasibility using simulation in an aerospace company. In a case study at Ericsson in Sweden [51], 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 explored current challenges involved in state-of-practice CIA, and propose five explicit improvements to address them, including two that we employ in this paper: 1) introducing a knowledge base of old CIA results, and 2) 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 three areas of IR in software engineering relevant to our work: *IR-based tracing* [27], *duplicate detection of issue reports.* [52], and *IR-based CIA* [53].

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 [54]. 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 such 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 [55]. 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 [56]. 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 a tool called *RETRO* [57]. De Lucia *et al.*'s research in this topic focuses on empirically evaluating LSI-based trace recovery in their document management system *ADAMS* [28]. 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* [58]. Much of their work has focused on improving the accuracy of their tool by various enhancements, e.g., project glossaries and term-based methods.

In a recent systematic mapping study of IR-based trace recovery [27], 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 [59]. Furthermore, most IR-based trace recovery evaluations do not involve humans, i.e., constitute *in silico* studies, and only one primary study was conducted as a case study in industry [26]. Finally, there appears to be little practical difference between different IR models [60], which corroborates previous observations by other researchers [23], [24], [61].

IR-based techniques were also used for supporting the CIA process. Arora *et al.* focused on inter-requirements CIA and suggested a normalized matching score based on propagation conditions. They implemented the approach in a prototype tool called NARCIA (NAtural language Requirements Change Impact Analyzer) and evaluated it in two industrial cases with 160 and 72 requirements, respectively [53]. Poshyvanyk *et al.* used IR methods to measure conceptual coupling between object-oriented software classes [62].

IR systems in software engineering were also applied for issue report duplicate detection. Previous research conducted on large projects shows that the fraction of issue duplicates can be between 10-30% and might cause considerable extra effort during triage [52], [63], [64]. 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 [30].

The RSSE we present in this paper builds on previous research on IR in software engineering. Similar to IR-based tracing tools, the goal is to identify trace links in databases containing a plethora of artifacts. As a central part of our tool is to identify similar issue reports from a history of evolution and maintenance projects, our work is also related to duplicate issue report detection.

2.3 RSSEs for Change Management and CIA

RSSEs differ from other software engineering tools in three ways [31]. 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 [65]. Hipikat relies on Mining Software Repositories (MSR) [66] 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 were mainly qualitative, and assessed how helpful Hipikat was when resolving a new issue report. The IR evaluation reported 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 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 can help newcomers perform comparably to more knowledgeable developers, i.e., to implement high quality corrective fixes under time pressure.

Another RSSE is *eRose* developed by Zimmermann *et al.* [67]. 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* [68] 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, comprising over 8,000 commits from eight OSS projects, the top-10 recommendations from eRose contained 33% of the true changes. In 34% of the cases eRose did not provide any recommendations. On the other hand, when eRose indeed provided recommendations, a correct source code file was 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 [69]. 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 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 was roughly 0.5 at recall 0.25, and for Eclipse, precision was 0.3 at recall 0.15. Ying *et al.* argued 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.

Canfora and Cerulo developed *Jimpa* [70], 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 resulted in the following precision-recall pairs: 0.20-0.40, 0.05-0.20, and 0.15-0.90, and the authors considered the outcome promising.

Gethers *et al.* developed *ImpactMiner*, a tool that combines textual similarity analysis, dynamic execution tracing, and MSR techniques [45]. 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.* presented 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 corresponded roughly to precision 0.10-0.15 and recall 0.20-0.35.

2.4 Related Work and Evolution of ImpRec

ImpRec packages our previous research efforts in IR as an RSSE for CIA, by combining work on consolidation of natural language requirements from multiple sources [71], duplicate detection of issue reports [52], [72], IR-based trace recovery [23], [27], and issue network analysis [73]. The overall idea of ImpRec, i.e., to reuse traceability captured in an issue tracker, was originally proposed in 2013 [73]. While the technical implementation details of ImpRec were reported in a recently published book chapter [30], this paper brings significant empirical evidence from a large-scale industrial case study. Section 4 presents an overview of the approach implemented in ImpRec.

The work by Čubranić *et al.* on Hipikat [65], [74] was central in making key decisions 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 book chapter [30].

Ali *et al.* developed Trustrace [75], a tool implementing an approach similar to ImpRec. Trustrace uses IR to identify candidate trace links between requirements and source code. Subsequently, the tool mines the issue repository and the code repository to identify already existing trace links; such links are considered to be of 'high trust'. The two sets of links are then combined using a web-trust model to create a final set of trustable trace links, i.e., the 'low-trust' IR links are discarded/reranked.

ImpRec and Trustrace share three aspects: 1) IR is used to identify potential trace links, 2) existing high-trust trace links are mined from software repositories, and 3) links previously created through a collaborative development effort are combined in a networked structure. The main conceptual difference is that ImpRec first establishes a trusted semantic network and then applies IR to identify low-trust links to starting points in the network, whereas Trustrace first uses IR to collect low-trust links and then uses complementary mining of trusted links to increase the accuracy of the tool output. A difference in the intended use of the tools is that ImpRec, in line with the RSSE definition (cf. Section 2.3), estimates whether a piece of information is valuable to a user in a specific task, i.e., ImpRec recommends potentially impacted software artifacts whenever a developer conducts a CIA. Trustrace on the other hand, is used to support a one-shot activity to recover a complete set of trace links.

Several studies proposed history mining to identify artifacts that tend to change together [45], [67], [69], [70]. Especially the studies by Canfora and Cerulo [70] and Gethers *et al.* [45] use techniques similar to ours, as they combine mining source code repositories with IR techniques. However, while these four studies address CIA, they solely focus on source code [35].

The main contribution of this study comes from the empirical evaluation on industrial scale artifacts. 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 [31]. While simulating the operation of an RSSE can provide the correctness of the recommendations, it does not reveal anything about the users' reactions. Three previous RSSE evaluations provide strength of evidence [76], in terms of realism, matching our study: Čubranić *et al.*'s study on Hipikat [65], Kersten and Murphy's evaluation of MyLyn [77], and Treude *et al.*'s recent evaluation of TaskNavigator [78]. However, our study is unique in its combination of a thorough *in silico* evaluation (larger sample of artifacts systematically studied) and a longitudinal *in situ* study in industry (>6 months).

3 Industrial Context Description

This section contains a general description of the case company and presents the CIA work task as it is currently conducted by the involved practitioners. 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 belongs to a large multinational company in the power and automation sector. We further describe the context structured in six context facets, according to the guidelines provided by Petersen and Wohlin [79].

Products. The products under development constitute an automation system that has evolved for decades, whose oldest running source code is from the 1980s. The system is safety-critical, governed by IEC 61511 [8], 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# and VB. The automation system contains both embedded systems and desktop applications, e.g., an IDE for developing control programs according to IEC 61131-3 [80]. The system is SIL2 safety-certified[2], according to IEC 61508 [7].

Processes. Projects follow a stage-gate iterative development process that is tailored to support the safety certification activities performed prior to release. 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,

2. Safety integrity level (SIL) is used in several safety standards to define a relative level of risk reduction provided by a safety function, from SIL1 to SIL4 [7].

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. The company has a history that stretches more than a century, and the organization has a mix of experiences and cultures. 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. Hundreds of engineers work in this context, in a globally distributed organization. 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. Typical teams consist of 8-12 developers.

Market. The product is released to a small market with only few players. Important customers in various market segments, e.g., oil & gas, or pulp & paper, sometimes initiated bespoke development with specific feature requests. The market strategy is to offer the best possible automation system for very large industrial plants in the process automation domain.

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* [12], 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. Providing an 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 [7]. 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

TABLE 1
The CIA template used in the case company, originally presented by Klevin [81]. 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?**

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 correctly introduced in the system.

Developers conduct CIAs regularly during development and maintenance projects, and the CIA activity requires much effort. During initial interviews, conducted to define the scope of the study, the developers reported that they on average conduct CIAs weekly or bi-weekly. 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 a few minutes to several hours.

Developers and safety engineers at the case company confirmed that it is difficult to identify how a change affects the software system under development. Example challenges highlighted during initial interviews 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. The developers have contrasting thoughts about CIAs, some are positive as they see it as a healthy sign in complex engineering, others are neutral and just see it as part of the job description, and yet another group has a negative connotation to CIA. While most developers agree that CIA is a mundane activity that requires extensive browsing of technical documentation, the strongest critics also considered the current format of CIA to be a far too heavy construct with little value, just done to comply with the organization's implementation of the IEC 61508 standard. The engineers' perspectives on CIA is further studied in a separate publication [82], including a mapping of connotation versus importance.

Currently there is little tool support available for CIA in the organization, as is the case for safety-critical software engineering in general [18]. 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. Note that not more than roughly a quarter of the issue reports have attached CIA reports, instead most issues are closed without changes that require formal CIA. We found several reasons for this, such as non-repeatable issues, duplicated issues, and changes deferred by the change control board. Also, issues not necessitating changes to any production code do not require CIA, including pure document updates and changes to test code.

Developers typically manage CIA reports through the simple web interface of the issue repository. 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 controlled input to the CIA template by providing formatting and basic sanity checks.

In this contect, the most important feature of any new CIA tool is to ensure high quality CIA reports. While it is also relevant to help engineers conduct CIA faster, cost reductions remain a secondary concern. Consequently, relevant to later trade-off discussions, effectiveness is more important than efficiency, i.e., quality is more important than time and cost savings.

4 Approach and ImpRec

This section presents an overview of our solution, as well as a brief description of the implementation of our ideas in the tool ImpRec. Note that ImpRec is not a novel contribution of this paper, instead we refer to our previous book chapter for a comprehensive description [30].

4.1 Traceability Reuse for Impact Analysis

Developers at the case company put extensive 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. Developers author CIA reports to comply with the safety process, but does rarely 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 process [82], 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 [14]. By extracting trace links from the semi-structured CIA reports to previously impacted artifacts (a.k.a. link mining [83]), we establish a *knowledge base* of historical impact.

ImpRec uses the established knowledge base 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

In the first step, 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 IDs of requirements and HW descriptions[3] have distinguished formats. Finally, we store all extracted traces (i.e., triplets of source artifact, target artifact, and trace link [54]) in a knowledge base represented by a semantic network [84].

The knowledge base contains more than 32,000 issue reports in its entirety, 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 either requirements, test specifications, HW 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 (5,066), verified-by (4,180), needs-update (1,660), impacts-HW (1,684), and trace links whose type could not be determined (1,435). In the CIA reports, trace links to non-code artifacts are typically specified on a document-level granularity; requirements being the exception, which instead are specified on an individual level. Finally, the knowledge base contains 22,636 related-to links between issue reports. For further details on the mining step, incl. descriptions of artifacts and trace links, we refer to the technical description of ImpRec in the RSSE book edited by Robillard *et al.* [30]. In Figure 1, we highlight that semantic information is available in the knowledge base, by presenting specified-by trace links in green and verified-by trace links in blue. The same coloring scheme is used also for the corresponding artifacts, i.e., requirements are green and test specifications are blue.

4.3 Recommending Potential Impact

When the knowledge base is established, recommendations are calculated in three steps as described in Borg and Runeson [30]: 1) identification of textually similar issue reports

3. HW descriptions include HW types expressed in VHDL, a language used in electronics design for integrated circuits and field-programmable gate arrays (FPGA). HW types are collected in HW libs.

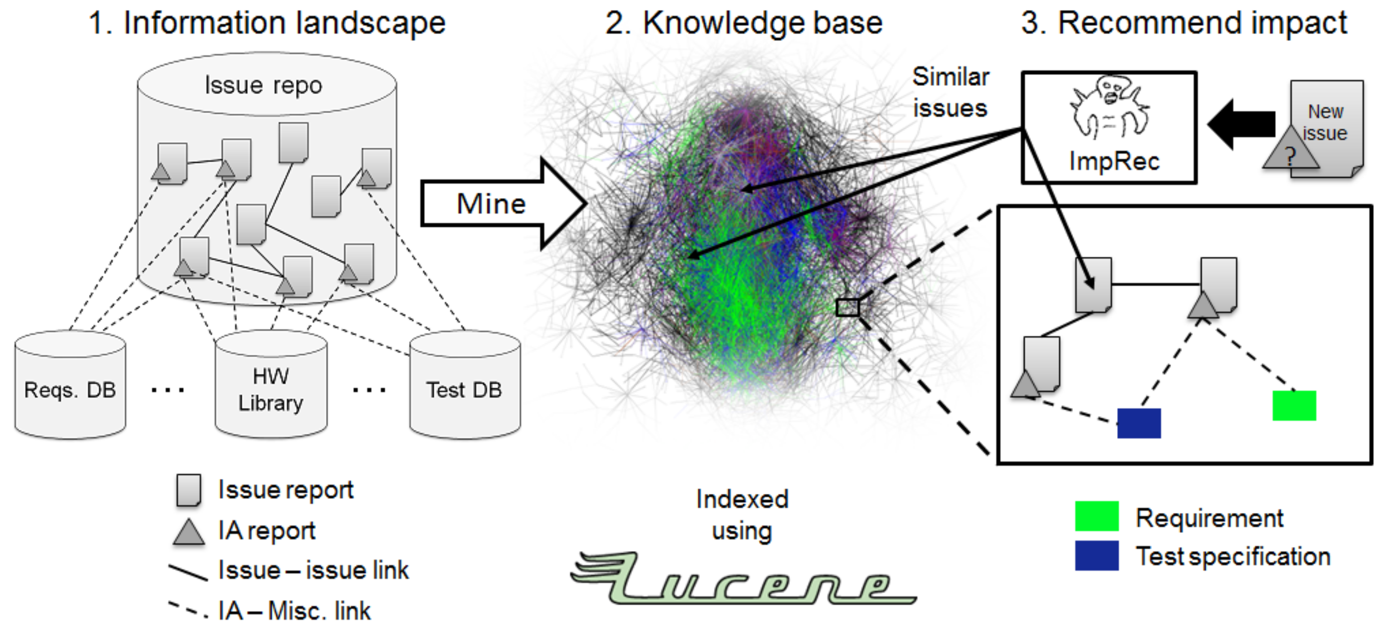


Fig. 1. Overview of the approach implemented in ImpRec. First, the issue repository is mined to establish a knowledge base. Second, ImpRec identifies similar issue reports in the knowledge base using Apache Lucene and recommends impact based on the network structure.

(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 [85], 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 initial step in the recommendation process is in line with previous work on duplicate detection of issue reports [72], IR-based trace capture [86], and content-based RSSEs [87].

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). This means that the artifacts that were reported as impacted in the starting points' previous CIA reports are considered candidate impact also for the current CIA. Then, related-to links are followed from the starting points to identify additional issue reports. For each new issue report found, its previously reported change impact is added to the impact set, and the search continues by following any further related-to links to unvisited issue reports. This second step is inspired by collaborative RSSEs [87], and helps the user to follow in the footprints (traces) of previous developers. The process of expanding the impact set by iteratively following related-to links is continued until a configurable number of links have been followed.

Third, we rank the artifacts in the impact sets[4]. 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, i.e., weights, to the ranking value for all n impact sets containing ART_x:

$$ranking\ value(ART_x) = \sum_{i=1}^{n} weight(ART_x \in SET_i) \quad (1)$$

and the weight of an artifact in an individual impact set is:

$$weight(ART_x \in SET_i) = \frac{\alpha * CENT_x + (1-\alpha) * SIM_x}{1 + LEVEL * PENALTY} \quad (2)$$

where $CENT_x$ is the centrality measure of ART_x in the knowledge base, SIM_x is the similarity of the issue report that was used as starting point when identifying ART_x, and $LEVEL$ is the number of related issue links followed from the starting point to identify ART_x. α and $PENALTY$ are constants that enable tuning for context-specific improvements, used to balance the importance of centrality vs. textual similarity and to penalize artifacts distant in the network, respectively. We cover context-specific tuning of ImpRec's parameters in detail in a parallel paper [88].

4.4 ImpRec: An RSSE for Change Impact Analysis

We implemented our approach in the prototype tool ImpRec[5]. 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. The first author spent in total more than a month on two development sites in Sweden and India.

4. For an illustrated example of the ranking process, we refer to Fig. 18.13 in the book chapter describing the ImpRec implementation [30].

5. 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 compared software engineering tool support to imps sitting on the shoulders of software developers to guide them [89].

An anonymized version of ImpRec, with some dummy data from the Android development project, is available on GitHub[6].

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, and 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 to denote whether the corresponding recommendation is relevant for the ongoing CIA. Every time a developer starts ImpRec, a unique session is created. During each 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/CancelRelevant – The developer toggles the check box of an item in the list view B or E.
- Confirmation – The developer clicks the 'Done' button (D), to conclude the feedback of the ongoing CIA task.

The user interface of ImpRec was designed with Murphy-Hill and Murphy's RSSE design factors in mind [90]. Murphy-Hill and Murphy argue that RSSE developers need to consider the following five factors: 1) distraction, 2) understandability, 3) assessability, 4) transparency, and 5) trust.

Distraction does not apply to ImpRec, as the users initiate all 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[7]. Moreover, to further support assessibility, when a user clicks on an item in list view B, the full description of the issue report is presented in area C, complemented by any stored CIA reports.

We argue that 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 list view B, and the result of the ImpRec ranking function in list view E). This decision is in contrast to general search tools, but it might help expert users to assess the value of the recommendations. Second, when the user selects issue reports in list view B, the items in list view 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.

6. https://github.com/mrksbrg/ImpRec
7. http://serg.cs.lth.se/research/experiment-packages/imprec

5 Research Method

In this section, we outline the evaluation part of this study and summarize the undertaken research method, see Figure 3 for an overview of the study.

Rationale and Purpose. This work was triggered by articulated needs at the case company associated with CIA for safety-critical development, and the potential solutions identified in the surveyed literature [27]. 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.

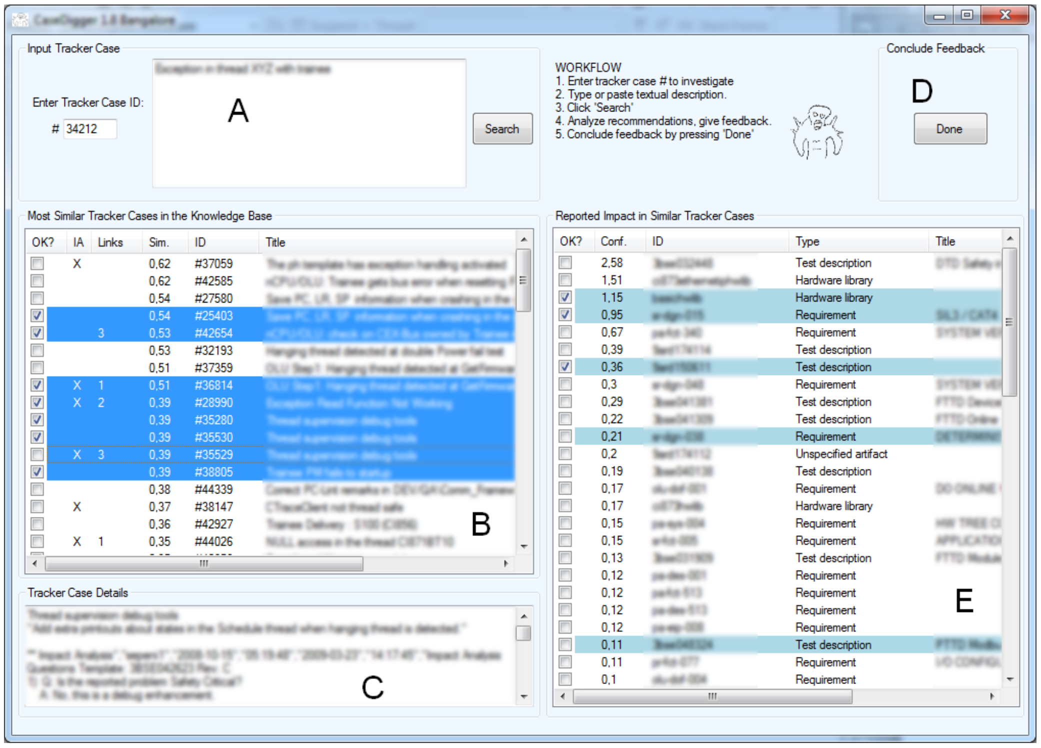
Since CIA involves both artifacts, people, and processes, as well as their interplay in a complex system development environment, it can be characterized as a "contemporary software engineering phenomenon within its real-life context, especially when the boundary between phenomenon and context cannot be clearly specified" [91], i.e., it is feasible for case study research.

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, 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.

Unit Sweden consisted of four developers in Sweden from the I/O team. Three developers volunteered during an initial presentation of the research, 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 [91]. To include yet another perspective in the study, we also asked the newest member of the I/O 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 consisted of seven developers in India from the Protocols team. The Protocols team was selected, after consultation with technical managers, since they share the same issue repository as Unit Sweden. As a consequence, the two units of analyses follow the same processes, which enables us to compare views from two continents. The seven developers were selected to cover as many perspectives from the Protocols team 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.* [32]. The first step of the development



Input Tracker Case

Enter Tracker Case ID:

34212

[illegible]

WORKFLOW
1. Enter tracker case # to investigate
2. Type or paste textual description.
3. Click 'Search'
4. Analyze recommendations, give feedback.
5. Conclude feedback by pressing 'Done'

Conclude Feedback

Most Similar Tracker Cases in the Knowledge Base

OK?	IA	Links	Sim.	ID	Title
	X		0,62	#37059	[illegible]
			0,62	#42585	[illegible]
			0,54	#27580	[illegible]
✓			0,54	#25403	[illegible]
✓		3	0,53	#42654	[illegible]
			0,53	#32193	[illegible]
			0,51	#37359	[illegible]
✓	X	1	0,51	#36814	[illegible]
✓	X	2	0,39	#28990	[illegible]
✓			0,39	#35280	[illegible]
✓			0,39	#35530	[illegible]
	X	3	0,39	#35529	[illegible]
✓			0,39	#38805	[illegible]
			0,38	#44339	[illegible]
	X		0,37	#38147	[illegible]
			0,36	#42927	[illegible]
	X	1	0,35	#44026	[illegible]

Tracker Case Details

[illegible]

Reported Impact in Similar Tracker Cases

OK?	Conf.	ID	Type	Title
	2,58	[illegible]	Test description	[illegible]
	1,51	[illegible]	Hardware library	
✓	1,15	[illegible]	Hardware library	
✓	0,95	[illegible]	Requirement	[illegible]
	0,67	[illegible]	Requirement	[illegible]
	0,39	[illegible]	Test description	
✓	0,36	[illegible]	Test description	
	0,3	[illegible]	Requirement	[illegible]
	0,29	[illegible]	Test description	[illegible]
	0,22	[illegible]	Test description	[illegible]
	0,21	[illegible]	Requirement	[illegible]
	0,2	[illegible]	Unspecified artifact	
	0,19	[illegible]	Test description	
	0,17	[illegible]	Requirement	[illegible]
	0,17	[illegible]	Hardware library	
	0,15	[illegible]	Requirement	[illegible]
	0,15	[illegible]	Requirement	[illegible]
	0,13	[illegible]	Test description	[illegible]
	0,12	[illegible]	Requirement	
	0,12	[illegible]	Requirement	
	0,12	[illegible]	Requirement	
	0,12	[illegible]	Requirement	
	0,11	[illegible]	Test description	[illegible]
	0,11	[illegible]	Requirement	[illegible]
	0,1	[illegible]	Requirement	

Fig. 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.

TABLE 2
Study participants in the dynamic validation. Junior/senior reflects years since the degree, whereas newcomer/seasoned depicts years of experience with the specific system.

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

involved extracting all history from the issue repository at the case company (A) and mining the semi-structured CIA reports to create a semantic network of software artifacts (B), as described in Section 4.2. Based on the semantic network, we iteratively developed ImpRec as described in Section 4.4.

The static validation was conducted using *in silico* simulation, in which ImpRec was evaluated on the historical inflow of issue reports (D), as proposed by Walker and Holmes [68]. 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.* [91]. We used the results from Unit Sweden 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 [14], [73]. ImpRec has continuously evolved from 2013 [30], 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) [88], and deployed the RSSE in Unit India (G) in August 2014. We conducted post-study interviews and concluded data collection in December 2014.

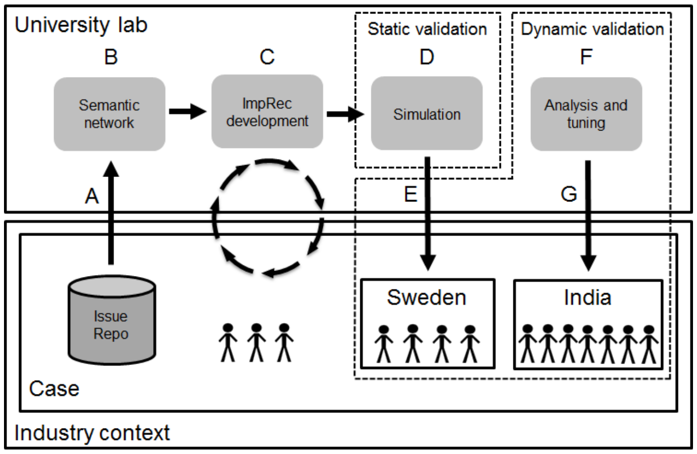


Fig. 3. Overview of study design. A: Data extraction from issue repository. B: Knowledge base established in the form of a semantic network. C: ImpRec iteratively developed with short feedback loops. D: Static validation based on simulation. E: ImpRec deployed in Team Sweden. F: Intermediate results analyzed and tuning of ImpRec [88]. G: Tuned ImpRec deployed in Team India.

5.1 Research Questions

The following research questions guided 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 accuracy discussions in line with previous work, but we also go beyond set-based measures by reporting Mean Average Precision (MAP) [92]. Section 5.4 presents further details on the measures. Note that while we address RQ1 quantitatively, direct comparisons of ImpRec and approaches from previous work on the same datasets are outside of the scope of this paper; the purpose of RQ1 is not to display how ImpRec stands in the competition, but rather to provide a feasibility test in preparation of RQ2 and RQ3.

We tackled RQ2 and RQ3 by deploying ImpRec in Unit Sweden and Unit India. 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? In order to assess the utility of ImpRec we carried out interviews with practitioners using the QUPER model [93] to relate the quality level in terms of recall to the perceived benefit.

The QUPER model was originally developed to support roadmapping of quality requirements, but it is here used as a basis for the interview instrument used in semi-structured interviews with developers. The QUPER model relates quality requirements, on e.g. software performance, to the market's acceptance in relation to three breakpoints of *utility* where the quality starts to be acceptable, *differentiation* where the quality starts to be a competitive advantage, and *saturation* where there is no economic value in further optimization. For more information on QUPER evaluation and usage guidelines, see [94] and [95].

RQ3 is an attempt to corroborate an assumption from previous research: newcomer developers benefit the most from RSSEs. Čubranić *et al.* developed Hipikat particularly to support developers new to OSS projects [74], and ten years later Panichella *et al.* also developed RSSEs for OSS project newcomers [96].

5.2 Static Validation: *In silico* Evaluation

We conducted a simulation study to initially justify our RSSE, as proposed by Walker and Holmes [68]. 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.* [97, pp. 248] , 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 [12], 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 [68, pp. 301]. 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. For the static validation, we used 26,120 chronologically ordered issue reports and their 4,845 attached CIA reports from the last 12 years of development, a dataset that we also studied in previous work [14]. To ensure a realistic simulation, we did not filter the dataset in any way.

We split the chronologically ordered data into a training set[8] (88%) and a test set (12%). 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, i.e., we study one set of 596 simulated CIA tasks. 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 [97] in the simulation was 80%, i.e., only 80% of the artifacts we wanted to link were available in the knowledge base.

We compared 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 parallel work [88]). ImpRec Text is a baseline that only relies on

8. We use the term "training set" although there is no supervised learning; the training set is used to establish the knowledge base, i.e., creating a semantic network and the corresponding Lucene index.

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. This baseline always reports the artifacts with the highest centrality measures, no matter the textual content of the incoming issue report, i.e., the same recommendations are presented regardless of input.

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 [31]. 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 a 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.

To better understand the operational context before deploying ImpRec, we conducted semi-structured interviews with all participants individually, and three additional senior engineers (A-C in Table 2). The interviews[9], roughly 45 min long, covered the state-of-practice CIA work task, and challenges experienced by the interviewees. 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 a 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 tutorial, 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 to use ImpRec 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. For convenience, feedback and reflections from Unit India were collected via email, although the rest of the study was conducted on site. The goal of the post-study interviews was to discuss utility focused on quality breakpoints as introduced in the QUPER model [93]. The QUPER model treats quality as an inherent characteristic on a non-linear sliding scale. 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 [11]. However, while they presented initial threshold based on their own experiences, we explore quality levels based on interviews with developers.

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 [27], or micro-evolution [67]).

Recall is the fraction of the true impact that ImpRec recommends (max 80% in the simulation, see Section 5.2). *Precision* is the fraction of the ImpRec recommendations that indeed represents 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 [28], [57], [98], 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 [99]. *Mean Average Precision* (MAP) is a secondary IR measure [100], 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 [101]. The measures 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

9. The interview guide is available on the companion website http://serg.cs.lth.se/research/experiment-packages/imprec/

within two weeks after the interview. We conducted thematic analysis [102] 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[10] contain rich information, and thus enable Search Log Analysis (SLA) [103]. 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 (list view B in Fig. 2) as there is no incentive for the participants to click on individual items among the potential impact (list view C in Fig. 2).

6 Results and Interpretation

This section presents the results from the static and dynamic validation, as well as our 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 simulating the inflow of issue reports. The simulation was based on 12 years of unfiltered issue reports from the case company, see Section 5.2 for details.

The four plots 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, detailed in Section 5.2: ImpRec A (solid line), ImpRec B (dashed line), ImpRec Text (dot-dashed line), and ImpRec Cent (dotted line). In Figure 4, the horizontal 'ceiling' line shows the highest possible recall for this dataset due to the catalog coverage, as described in Section 5.2.

Figure 4 a) presents how the recall improves as N increases. ImpRec A is the best configuration before N exceeds 10 (Rc@5=0.33 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. A possible interpretation is that the low initial recall (Rc@1-9<0.20) of the purely textual baseline 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. Note that as both ImpRec A and B rely on textual similarity to

10. The log files were collected after consent from the participants.

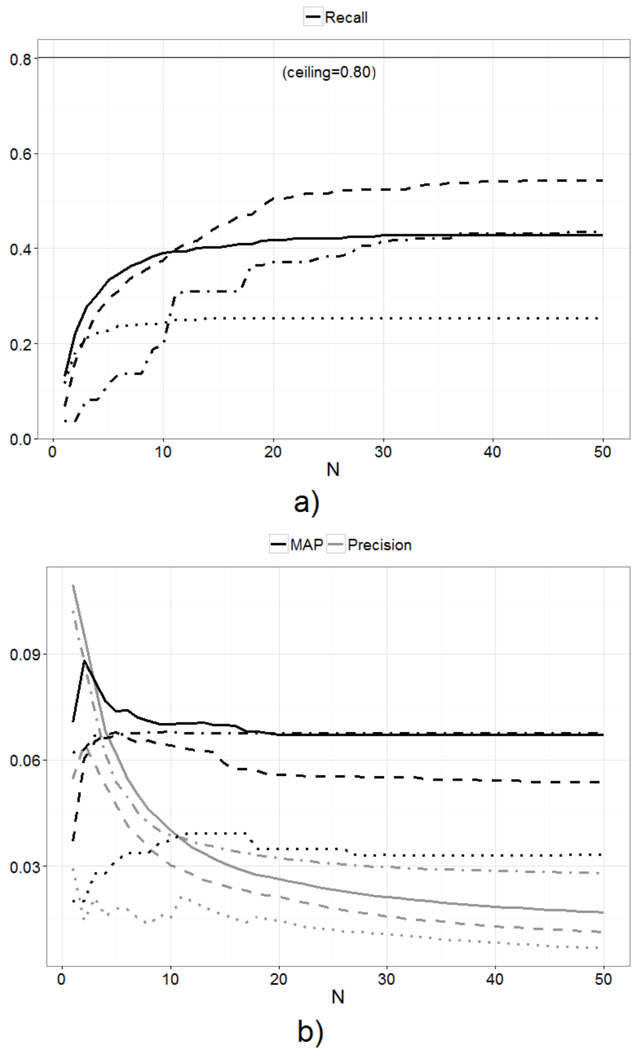
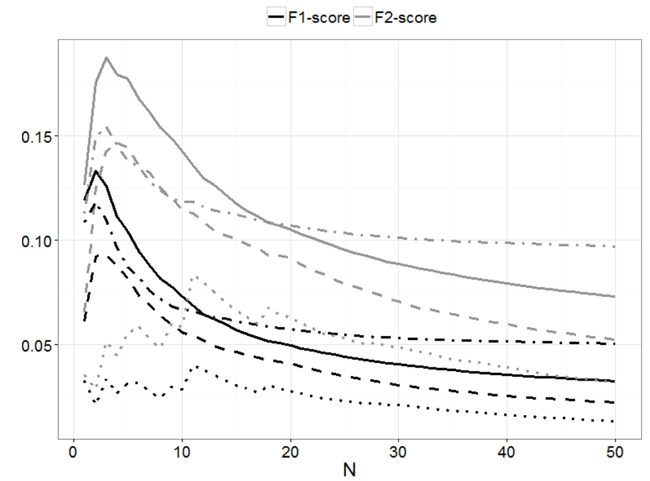


Fig. 4. Recall, precision, and MAP from the static validation, i.e., the *in silico* simulation. Solid line: ImpRec A, dashed line: ImpRec B, dot-dashed line: ImpRec Text, and dotted line: ImpRec Cent.

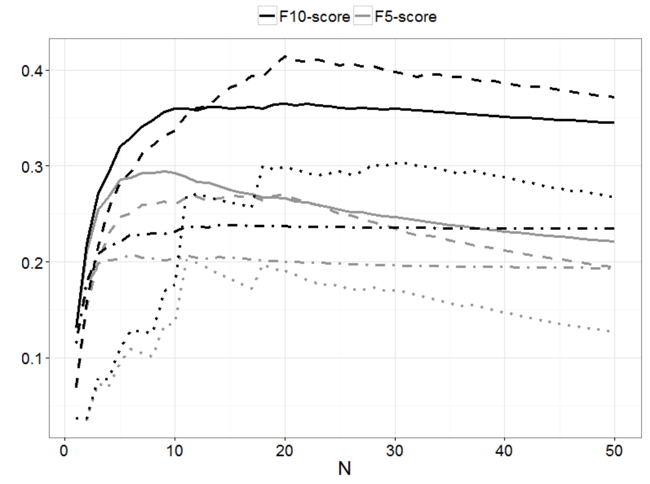
identify starting points, their recall values also converge at recall levels lower than the ceiling. Finally, ImpRec Cent (dotted line) displays a notably worse recall curve.

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 a) 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



a)



b)

Fig. 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.

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 b) 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 (N=9) from ImpRec A and twenty recommendations (N=20) 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 observe 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

TABLE 3
ImpRec correctness compared to previous work. @10 refers to a cut-off point of 10 recommendations, @? means the cut-off point was not reported.

	Study	**Pr**	**Rc**	**Project(s)**
@10	ImpRec	0.05	0.40	Automation
	Canfora and Cerulo [70]	0.20	0.40	Gedit
		0.05	0.20	ArgoUML
		0.15	0.90	Firefox
	Gethers *et al.* [104]	0.15	0.20	ArgoUML
		0.10	0.20	JabRef
		0.15	0.35	jEdit
		0.10	0.25	muCommander
	Zimmermann *et al.* [67]		0.33 (avg.)	Eclipse, gcc, Gimp, JBoss, jEdit, KOffice, Postgres, Python
@?	Čubranić *et al.* [65]	0.10	0.65	Eclipse
	Ying *et al.* [69]	0.5	0.25	Mozilla

candidates.

Table 3 lists the correctness of ImpRec compared to the related work presented in Section 2.3. While there are considerable differences between the studies, the results indicate that ImpRec performs in line with previous work on CIA. This result mirrors our aim to reach high recall within a reasonable (browsable) amount of recommendations. Note that the low Pr@10 is inevitable due to the nature of our test set. Our test set (i.e., gold standard) contains 596 issue reports (i.e., queries) and only 320 true links (i.e., relevant documents), thus many queries have no matching true documents. Thus, the highest possible precision for Pr@1 and Pr@10 are 320/596=0.54 and 320/5,960=0.054, respectively.

To summarize, the static validation results in correctness, i.e., precision and recall measures, in line with previous work, even though ImpRec addresses heterogeneous non-code artifacts rather than source code. Only the approach by Canfora and Cerulo (when evaluated on the Firefox project) outperforms ImpRec for Rc@10, obtaining a value as high as 0.90. At the same time, ImpRec's precision is in the lower end among the other work. However, as ImpRec obtains a recall of 0.4 already for 10 recommendations (Rc@10=0.4), we do not consider the low precision to be a major problem, i.e., a developer would not have to sift through pages of artifacts to find actionable recommendations. Thus, we consider the correctness of ImpRec to be good enough to initiate the dynamic validation and turn our attention to RQ2 and RQ3.

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.

6.2.1 Overview of the Results

The initial interviews confirmed the importance of supporting CIA, and that considerable effort is spent. The participants estimated that developers on average work 50-100 hours on CIA yearly [82], but providing precise numbers is difficult; there is no consensus of what should be included in CIA as it cannot easily be separated from activities such as program understanding, software testing, and debugging. The interviewees reported that the frequency of the CIAs depends on the phase of the project, but one CIA per week with 1-2 hours effort appeared to be the average for most developers. However, the interviews refuted the value of the two measures TimeCIA and ModCIA defined in Section 5.3, as too confounded by other variables[11], and only useful as "a very rough indication of effort and complexity" (participant A) Thus, we focused the dynamic validation on Search Log Analysis (SLA) combined with qualitative feedback from the post-interviews.

We received positive responses during initial installation and demonstration of ImpRec, with several participants expressing interest in trying it. One participant immediately found helpful recommendations during the demonstration, saying "this [issue report] was exactly what I was looking for actually" (participant 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 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, derived from the usage logs. First, the five leftmost 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 roughly five minutes per session with ImpRec. The SLA revealed that three 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 of the ImpRec recommendations per participant. 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 (denoted #Extra), and the corresponding result in terms of recall (in parenthesis: the maximum recall based on the knowledge

11. For a longer discussion on the challenges of measuring CIA, also based on the initial interviews, we refer to a separate publication [82].

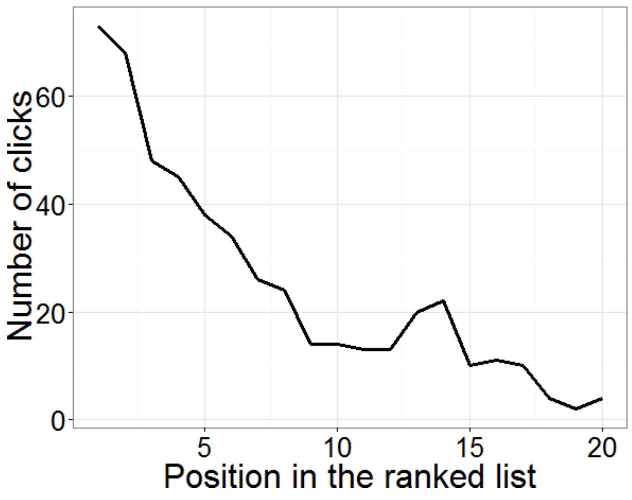


Fig. 6. Click distribution of the participants' interaction with recommended related issue reports.

base coverage). Finally, four columns show results regarding impacted artifacts, analogous to the related issues.

The participants used the confirmation clicks to report that ImpRec provided relevant items in 30 of the 43 ImpRec sessions (70%). Useful related issue reports were provided for 21 of 43 of the sessions (49%), and true recommendations in 14 of 31 (45%) of the sessions with a corresponding CIA report. In total, participants confirmed impacted items as relevant for 23 of the 43 sessions (54%). 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). For further details on the individual ImpRec sessions, we refer to the companion website[12].

Among the 50 true recommendations, the participants missed 19 of them (38%). As expected, the position of recommendations on the ranked lists was important. The click distribution of the 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 [105]. 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 also in previous work [73], underestimates the issue interrelations.

Regarding potentially impacted artifacts, the analysis of confirmation clicks shows that ImpRec presented 77 relevant impact recommendations, and 29 of them (38%)

12. http://serg.cs.lth.se/research/experiment-packages/imprec/

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)
Sweden	E	14 (14)	D: 101/501	9 min	23 (9)	6	5	0.26 (0.39)	62 (56)	28	8	0.45 (0.90)
Sweden	F	8 (4)	T: 9/47	8 min	10 (4)	2	13	0.2 (0.4)	33 (31)	14	7	0.42 (0.94)
Sweden	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)	1	0	0.25 (0.25)
India	I	1 (0)	T: 11/63	3 min	16 (6)	0	2	0 (0.38)	N/A	N/A	0	N/A
India	J	2 (1)	U: 2/15	2 min	4 (4)	0	5	0 (1)	2 (1)	0	1	0 (0.5)
India	K	1 (1)	U: 4/25	80 min	2 (0)	0	7	0 (0)	8 (0)	0	0	0 (0)
India	L	Performed no change impact analyses during the study										
India	M	1 (1)	U: 2/8	?	4 (4)	0	6	0 (1)	1 (1)	1	0	1 (1)
India	N	2 (2)	U: 3/14	6 min	1 (1)	0	5	0 (1)	0 (0)	0	6	N/A
Sum		**43 (31)**										

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.

6.2.2 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 in rapid succession. 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, participant D 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, his results are in line with the static validation. During the post-study interview, participant 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 [31] (the other goal is to provide novelty), and it is also critical in building the users' trust in the tool [97].

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, the tool 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", possibly decreasing her motivation to work with ImpRec.

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 the participant 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, 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 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 product management perspective, a type of user that browses rather than writes CIA reports. He confirmed that ImpRec delivered several useful

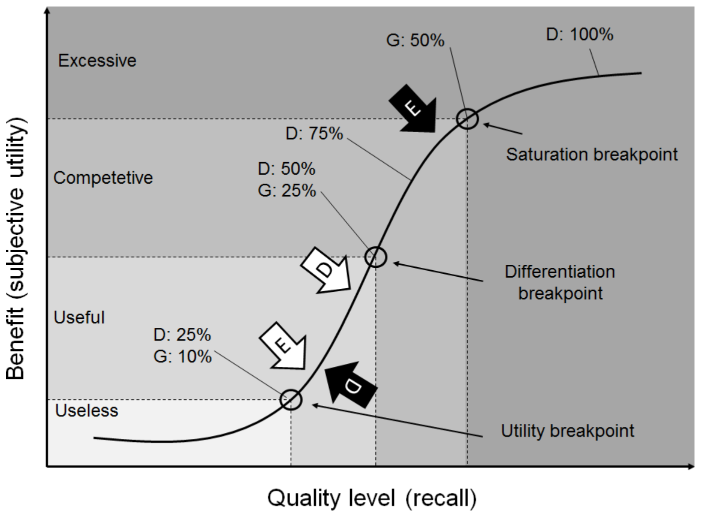


Fig. 7. Mapping correctness and utility using the QUPER model. Black arrows depict the current manual work, white arrows represent working with ImpRec.

recommendations (five related issue reports and six 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.

6.2.3 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 utility during post-study interviews with the participants in Unit Sweden, based on the QUPER model [93]. Figure 7 shows an overview of the discussions. Three out of four developers found it useful to discuss quality versus 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 has 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 significance. 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 on the benefit dimension. 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"'. Despite his critical remark, he acknowledged that other roles than developers might find them more useful, especially toward the end of projects. He also made a suggestion: "CIA reports are 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." Still, he emphasized the variation of how skillful developers are at writing CIA reports, and that some tend to report too much. 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 think for themselves").

Participant F preferred to discuss utility without framing it into 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 useful recommendations, but particularly commended the fast search results in the tool. He also encouraged future evolution 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."

The 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 it reach its full potential. He also mentioned two additional improvement proposals: 1) filtering of the search results and 2) personalization of the searches, e.g., by tagging 'favorites'. 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.

Participants K and M from Unit India also provided positive qualitative feedback on the utility of ImpRec, however not structured according to the QUPER model. Some participants in Unit India were also eager to further try ImpRec, 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, see Section 9.

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.* [91]. We minimize conclusion validity discussion as the conclusions of this paper do not arrive from inferential statistics and its assumptions [106].

7.1 Static Validation (RQ1)

The main threats to our conclusions regarding correctness regard 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 under study, 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 relative to the gold standard. The gold standard is extracted from manually created CIA reports, 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 CIA reports during the development life-cycle.

To increase 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 operation, 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 research design lead to some simplifications of the static validation. Even though we simulated 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 [107], a type of k-fold cross validation preserving the internal order of elements. However, the extent of our static validation is in line with 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. Finally, 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 used correctness as a proxy for utility, and leave studies with real users as important future work.

7.2 Dynamic Validation (RQ2 and RQ3)

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 threats, the interview instruments were reviewed by all authors, and the interviewees were all given opportunity to openly discuss their experiences. 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 (check boxes in ImpRec), 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, referred to as evaluation apprehension by Wohlin *et al.* [106], we presented the location of the log file, and explained that all data were 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 calculated the measures per participant prior to the initial interviews. The interviewees explained that there were too many confounding factors to interpret the two measures; we thus did not attempt to triangulate the utility of ImpRec with these quantitative measures. Consequently, despite conducting a study in an industrial context, we do not report any measures of cost or time savings. Nonetheless, we argue that conducting interviews supported by the QUPER framework complemented by quantitative 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 best 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.

Another threat to the internal validity of the utility evaluation is the three different search strategies identified by the SLA. It is possible that ImpRec is more likely to be considered useful if users follow a particular search strategy. We notice that the three participants D, J, and K, who arguably provided the most positive qualitative feedback, all used ImpRec with manually crafted queries as input. While this observation suggests that free exploratory ImpRec sessions are the most fruitful, participants H and M reported less positive feedback using the same search strategy. Future work should further investigate different search strategies; in the meantime we recommend ImpRec users to try a combination of search strategies, preferably in an iterative fashion.

RQ3 addresses differences between project newcomers and seasoned developers. Since all seasoned developers are also senior, there is a risk that seniors have a more conservative 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 [108]. 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.

In contrast to the static validation, we discuss the external validity of the dynamic validation 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 had 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 from ImpRec as well.

Regarding generalization to other companies, analytical generalization is required to relate our findings to other cases [91]. 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. Other organizations developing safety-critical systems may have other adaptations of their development processes to fulfill safety standards such as IEC 61511 [8], ISO 26262 [9], and EN 50128 [11]. Still, explicit CIAs are required in all organizations modifying a safety certified software system [4]. Moreover, our cross-domain survey of CIA also suggests that while details differ, many of the CIA challenges are universal in safety-critical development [18]. 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 [97], but this study focused only on correctness and utility. Other researchers could have selected other attributes. From our perspective, the most important complementing attributes to study in future work are coverage and risk, as discussed in Section 9.

8 Summary and 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. Finally, we relate our experiences to previously reported experiences of case study research in software engineering.

8.1 Revisiting the Research Questions

RQ1 addresses the correctness of ImpRec recommendations [97], a question that we tackled using *in silico* simulation and quantitative IR measures, in what we refer to as static validation [32]. 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.

ImpRec is different from the majority of tools providing automated CIA by its explicit focus on non-code software artifacts. While this difference 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 [28], [57], [98] and consider recall more important (as long as the total number of recommendations is still reasonable). Thus, we postulate that *the correctness of ImpRec is good enough to commence dynamic validation.*

RQ2 deals with the utility of ImpRec [97], and especially whether developers recognized the value of the RSSE. We assessed this using dynamic validation [32], 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 [93].

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 did 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., reinforcing what the developers already know, but providing limited novelty [31]. On the other hand, delivering confirmation is critical for an RSSE to establish user trust [97]. 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 [109], but as the organization was (and still is) in a transition to replace the current issue tracker, we decided to instead integrate ImpRec in IA Sidekick, an existing support tool. However, as most developers were unaware of IA Sidekick, it did not support the dissemination of ImpRec as much as expected.

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 [65], [96]. 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

TABLE 5
ImpRec compared to Lehnert's taxonomy of CIA tools [35]. 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	 ≈ 50,000 entities Pr@10=0.05 Rc@10=0.40 < 1 s per search + *in situ evaluation*

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 might help newcomers*.

The seasoned developers expressed particular 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, 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 RSSE 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 compare our work to previous CIA research based on the taxonomy of CIA support proposed by Lehnert [36] and his accompanying literature review [35], see Table 5.

First, although CIA is fundamental in safety standards (e.g., [9], [10], [11]), most of the previous evaluations of tool support for CIA exclusively consider Open Source Software (OSS) [35]. We have previously identified the same lack of studies in proprietary contexts regarding tool support for traceability management [27], another cornerstone in safety-critical development [110], [111].

Exclusively focusing on OSS is unfortunate, since there is a need for empirical studies on safety-critical software development in industry [13]. Also, there are still no large safety-critical OSS systems available as surrogates [112], thus researchers must target proprietary systems. The over-representation of studies in the OSS domain applies to software engineering research in general, partly explained by the appealing availability of large amounts of data and the possibility of replications [113]. Thus, this study is a rare example of an in-depth empirical study on tool support for CIA in a proprietary context.

Second, we conducted a longitudinal *in situ* study. Most previous evaluations on CIA tools were 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 longitudinal analysis of a deployed tool [35]. In the RSSE community, Robillard and Walker recently highlighted the lack of studies that include humans in the loop [31]. Our study is unique in its *in situ* design. 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. Our work is one of the few exceptions that deal with miscellaneous artifacts. In safety-critical software development, the impact on different types of artifacts must be analyzed, which is confirmed by our previous survey [18] and the initial interviews in the current study. 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 target industrial software systems certified by some of the established safety standards, to enable additional 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 [36]. 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 [114], as well as HM+IR [70], [115], [116] and TR+IR [117], but no other CIA tool combines HM+TR+IR. Instead, the most similar tools from previous work, both combining HM+TR+IR, are Hipikat [65] and Trustrace [75]. While neither Hipikat nor Trustrace were developed explicitly for CIA (supporting issue resolution and trace recovery, respectively; thus not included by Lehnert), both tools could also support CIA.

Regarding the remaining criteria in Lehnert's taxonomy [36], 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

[35]. 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 complex 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 is a complex cognitive task. The initial interviews in this study clearly showed that assessing the value of tool support cannot be made using simple measures such as TimeCIA and ModCIA (discussed in Section 5.3), due to the many confounding factors [82]. Furthermore, our post-study interviews revealed 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 overall*. 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 [118]. 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*. 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 mining explicit knowledge from historical CIA reports. As presented in Section 3.2, one of the challenges of a 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 confirmed 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 tend to take pride in evolving source code to high standards, but the CIA report is primarily a one-shot production, and developers rarely look back.

Albeit 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. Also, ImpRec can only recommend already encountered artifacts in the collaboratively created knowledge base. This aspect is referred to as the RSSE *coverage* [97]. 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 publication [119], in which they categorize the challenges along three dimensions: 1) technical, 2) collaboration, and 3) motivational.

Technical challenges originate in the complex 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 usage 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 [119], we created an easy-to-follow user manual and provided support via email. The *motivational dimension* is probably the most protruding among the 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.

We identified an additional dimension not mentioned by Dubey and Hudepohl [119], that 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, a kind of CIA for the tool chain. 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.

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 deployed ImpRec [30] 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 incoming issue reports, and subsequently uses network analysis in a collaboratively 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.* [32]. 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 assessed the utility of ImpRec (RQ2) based on collected user logs and interviews. Our results indicate that ImpRec's current level of correctness 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. On the other hand, we conclude that the quality of the ImpRec recommendations is not high enough to replace human engineers, but they can be used for CIA validation and to support project newcomers (RQ3). A CIA validation tool could be highly valuable in a safety-critical development context. Any true impact recommended by a tool, previously missed by humans, would be of great importance – typically more so than CIA cost savings.

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. Also, our work focuses on CIA support for non-code software artifacts, an understudied 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, could be a waste of effort. By conducting link mining, the knowledge in the historical CIA reports can be reused to provide decision support that might be especially helpful for junior engineers and new employees.

While the current version of ImpRec appears to be mature enough to be used in industry, there are several important avenues 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. Yet another idea is to combine the semantic network of ImpRec with the trust model implemented in Trustrace, an amalgamation that could improve ImpRec's ranking process.

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 [120].

Finally, research must be directed at how to introduce additional tool support in safety-critical contexts, in line with work by Dupey and Hudepohl [119]. 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[13], and also partially supported by a research grant for the ORION project (reference number 20140218) from The Knowledge Foundation in Sweden. Thanks go to all participants in the case study.

References

[1] 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*, 2014, pp. 125–134.

[2] 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*, 2010, pp. 412–420.

[3] 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*, 2011, pp. 746–755.

[4] S. Bohner, "Software Change Impacts - An Evolving Perspective," in *Proc. of the 18th International Conference on Software Maintenance*, 2002, pp. 263–272.

[5] K. Chen and V. Rajlich, "RIPPLES: Tool for Change in Legacy Software," in *Proc. of the 17th International Conference on Software Maintenance*, 2001, pp. 230–239.

[6] 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*, 2010, pp. 14–22.

[7] International Electrotechnical Commission, *IEC 61508 ed 1.0, Electrical/Electronic/Programmable Electronic Safety-Related Systems*, 2010.

[8] ——, *IEC 61511-1 ed 1.0, Safety Instrumented Systems for the Process Industry Sector*, 2003.

[9] International Organization for Standardization, *ISO 26262-1:2011 Road Vehicles - Functional Safety*, 2011.

[10] Radio Technical Commission for Aeronautics, "DO-178C Software Considerations in Airborne Systems and Equipment Certification," Tech. Rep., 2012.

[11] European Committee for Electrotechnical Standardisation, *Railway Applications - Safety Related Electronic Systems for Signaling*, 1999.

[12] T. Kelly, "Arguing Safety - A Systematic Approach to Managing Safety Cases," PhD Thesis, University of York, 1999.

[13] 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*, vol. 56, no. 7, pp. 689–717, 2014.

[14] 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.

[15] I. Chou, "Secure Software Configuration Management Processes for Nuclear Safety Software Development Environment," *Annals of Nuclear Energy*, vol. 38, no. 10, pp. 2174–2179, 2011.

[16] K. Grimm, "Software Technology in an Automotive Company: Major Challenges," in *Proc. of the 25th International Conference on Software Engineering*, 2003, pp. 498–503.

13. http://ease.cs.lth.se

[17] 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*, 2009, pp. 205–216.

[18] J. de la Vara, M. Borg, K. Wnuk, and L. Moonen, “An Industrial Survey of Safety Evidence Change Impact Analysis Practice,” *IEEE Transactions on Software Engineering (To appear)*, 2016.

[19] 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*, 2014, pp. 40–49.

[20] L. Briand, Y. Labiche, L. O’Sullivan, and M. Sówka, “Automated Impact Analysis of UML Models,” *Journal of Systems and Software*, vol. 79, no. 3, pp. 339–352, 2006.

[21] J. Cleland-Huang, W. Marrero, and B. Berenbach, “Goal-Centric Traceability: Using Virtual Plumblines to Maintain Critical Systemic Qualities,” *Transactions on Software Engineering*, vol. 34, no. 5, pp. 685–699, 2008.

[22] 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*, 2003, pp. 128–137.

[23] 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*, 2011, pp. 27–34.

[24] 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*, 2010, pp. 231–240.

[25] A. De Lucia, R. Oliveto, and G. Tortora, “Assessing IR-Based Traceability Recovery Tools Through Controlled Experiments,” *Empirical Software Engineering*, vol. 14, no. 1, pp. 57–92, 2009.

[26] 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*, 2008, pp. 100–111.

[27] M. Borg, P. Runeson, and A. Ardö, “Recovering from a Decade: A Systematic Mapping of Information Retrieval Approaches to Software Traceability,” *Empirical Software Engineering*, vol. 19, no. 6, pp. 1565–1616, 2014.

[28] 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*, vol. 16, no. 4:13, 2007.

[29] 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*, vol. 3, no. 3, pp. 193–202, 2007.

[30] M. Borg and P. Runeson, “Changes, Evolution and Bugs - Recommendation Systems for Issue Management,” in *Recommendation Systems in Software Engineering*, M. Robillard, W. Maalej, R. Walker, and T. Zimmermann, Eds. Springer, 2014, pp. 477–509.

[31] M. Robillard and R. Walker, “An Introduction to Recommendation Systems in Software Engineering,” in *Recommendation Systems in Software Engineering*, M. Robillard, W. Maalej, R. Walker, and T. Zimmermann, Eds. Springer, 2014, pp. 1–11.

[32] T. Gorschek, C. Wohlin, P. Carre, and S. Larsson, “A Model for Technology Transfer in Practice,” *IEEE Software*, vol. 23, no. 6, pp. 88–95, 2006.

[33] E. Engström, P. Runeson, and M. Skoglund, “A Systematic Review on Regression Test Selection Techniques,” *Information and Software Technology*, vol. 52, no. 1, pp. 14–30, 2010.

[34] S. Bohner, *Software Change Impact Analysis*. IEEE Computer Society Press, 1996. [Online]. Available: http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.117.6777

[35] S. Lehnert, “A Review of Software Change Impact Analysis,” Ilmenau University of Technology, Tech. Rep., 2011. [Online]. Available: http://nbn-resolving.de/urn:nbn:de:gbv:ilm1-2011200618

[36] ——, “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*, 2011, pp. 41–50.

[37] R. Arnold and S. Bohner, “Impact Analysis - Towards a Framework for Comparison,” in *Proc. of the 9th Conference on Software Maintenance*, 1993, pp. 292–301.

[38] M. Abi-Antoun, Y. Wang, E. Khalaj, A. Giang, and V. Rajlich, “Impact Analysis Based on a Global Hierarchical Object Graph,” in *Proc. of the 22nd International Conference on Software Analysis, Evolution, and Reengineering*, 2015, pp. 221–230.

[39] M. Petrenko and V. Rajlich, “Variable Granularity for Improving Precision of Impact Analysis,” in *Proc. of the 17th International Conference on Program Comprehension*, 2009, pp. 10–19.

[40] J. Law and G. Rothermel, “Whole Program Path-Based Dynamic Impact Analysis,” in *Proc. of the 25th International Conference on Software Engineering*, 2003, pp. 308–318.

[41] A. Orso, T. Apiwattanapong, J. Law, G. Rothermel, and M. J. Harrold, “An Empirical Comparison of Dynamic Impact Analysis Algorithms,” in *Proc. of the 26th International Conference on Software Engineering*, 2004, pp. 491–500.

[42] D. Amyot, “Introduction to the User Requirements Notation: Learning by Example,” *Computer Networks*, vol. 42, no. 3, pp. 285–301, 2003.

[43] J. Cleland-Huang, R. Settimi, O. BenKhadra, E. Berezhanskaya, and S. Christina, “Goal-centric Traceability for Managing Non-functional Requirements,” in *Proc. of the 27th International Conference on Software Engineering*, 2005, pp. 362–371.

[44] B. Li, X. Sun, H. Leung, and S. Zhang, “A Survey of Code-Based Change Impact Analysis Techniques,” *Software Testing, Verification and Reliability*, vol. 23, no. 8, pp. 613–646, 2013.

[45] 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*, 2012, pp. 430–440.

[46] S. Ghosh, S. Ramaswamy, and R. Jetley, “Towards Requirements Change Decision Support,” in *Proc. of the 20th Asia-Pacific Software Engineering Conference*, 2013, pp. 148–155.

[47] 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*, 2014, pp. 444–454.

[48] 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.

[49] 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*, 2012, pp. 355–360.

[50] M. Kilpinen, “The Emergence of Change at the Systems Engineering and Software Design Interface,” PhD Thesis, University of Cambridge, 2008.

[51] P. Rovegård, L. Angelis, and C. Wohlin, “An Empirical Study on Views of Importance of Change Impact Analysis Issues,” *Transactions on Software Engineering*, vol. 34, no. 4, pp. 516–530, 2008.

[52] 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*, 2007, pp. 499–510.

[53] C. Arora, M. Sabetzadeh, A. Goknil, L. C. Briand, and F. Zimmer, “Change Impact Analysis for Natural Language Requirements: An NLP Approach,” in *Proc. of the 23rd International Requirements Engineering Conference*, Aug 2015, pp. 6–15.

[54] 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 *Software and Systems Traceability*, J. Cleland-Huang, O. Gotel, and A. Zisman, Eds. Springer, 2012, pp. 3–22.

[55] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, “Recovering Traceability Links between Code and Documentation,” *Transactions on Software Engineering*, vol. 28, no. 4, pp. 970–983, 2002.

[56] 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*, vol. 15, no. 5, pp. 811–836, 2005.

[57] J. Huffman Hayes, A. Dekhtyar, and S. Sundaram, “Advancing Candidate Link Generation for Requirements Tracing: The Study of Methods,” *Transactions on Software Engineering*, vol. 32, no. 1, pp. 4–19, 2006.

[58] X. Zou, R. Settimi, and J. Cleland-Huang, “Improving Automated Requirements Trace Retrieval: A Study of Term-Based Enhance-

ment Methods," *Empirical Software Engineering*, vol. 15, no. 2, pp. 119–146, 2010.

[59] 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*, 2012, pp. 181–190.

[60] 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*, 2013, pp. 243–246.

[61] 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*, 2010, pp. 68–71.

[62] D. Poshyvanyk, A. Marcus, R. Ferenc, and T. Gyimóthy, "Using Information Retrieval Based Coupling Measures for Impact Analysis," *Empirical Software Engineering*, vol. 14, no. 1, pp. 5–32, 2009.

[63] J. Anvik, L. Hiew, and G. Murphy, "Coping with an Open Bug Repository," in *Proc. of the 2005 OOPSLA Workshop on Eclipse Technology eXchange*, 2005, pp. 35–39.

[64] N. Jalbert and W. Weimer, "Automated Duplicate Detection for Bug Tracking Systems," in *Proc. of the 38th International Conference on Dependable Systems and Networks*, 2008, pp. 52–61.

[65] D. Cubranić, G. Murphy, J. Singer, and K. Booth, "Hipikat: A Project Memory for Software Development," *Transactions on Software Engineering*, vol. 31, no. 6, pp. 446–465, 2005.

[66] 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*, vol. 19, no. 2, pp. 77–131, 2007.

[67] 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*, 2004, pp. 563 – 572.

[68] R. Walker and R. Holmes, "Simulation - A Methodology to Evaluate Recommendation Systems in Software Engineering," in *Recommendation Systems in Software Engineering*, M. Robillard, W. Maalej, R. Walker, and T. Zimmermann, Eds. Springer, 2014, pp. 301–327.

[69] A. Ying, G. Murphy, R. Ng, and M. Chu-Carroll, "Predicting Source Code Changes by Mining Change History," *Transactions on Software Engineering*, vol. 30, no. 9, pp. 574–586, 2004.

[70] 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*, 2006, pp. 105–111.

[71] J. Natt och Dag, V. Gervasi, S. Brinkkemper, and B. Regnell, "A Linguistic-Engineering Approach to Large-Scale Requirements Management," *IEEE Software*, vol. 22, no. 1, pp. 32–39, 2005.

[72] 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.

[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*, 2013, pp. 79–88.

[74] D. Cubranić, "Project History as a Group Memory: Learning From the Past," PhD Thesis, University of British Columbia, 2004.

[75] N. Ali, Y. Guéhéneuc, and G. Antoniol, "Trustrace: Mining software repositories to improve the accuracy of requirement traceability links," *Transactions on Software Engineering*, vol. 39, no. 5, pp. 725–741, 2013.

[76] 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*, 2008, pp. 178–187.

[77] M. Kersten and G. Murphy, "Using Task Context to Improve Programmer Productivity," in *Proc. of the 14th International Symposium on Foundations of Software Engineering*, 2006, pp. 1–11.

[78] C. Treude, M. Robillard, and B. Dagenais, "Extracting development tasks to navigate software documentation," *Transactions on Software Engineering*, vol. 41, no. 6, pp. 565–581, June 2015.

[79] K. Petersen and C. Wohlin, "Context in Industrial Software Engineering Research," in *Proc. of the 3rd International Symposium on Empirical Software Engineering and Measurement*, 2009, pp. 401–404.

[80] International Electrotechnical Commission, *IEC 61131-3 ed 3.0, Programmable Controllers - Part 3: Programming Languages*, 2013.

[81] 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.

[82] M. Borg, J.-L. de la Vara, and K. Wnuk, "Practitioners' Perspectives on Change Impact Analysis for Safety-Critical Software - A Preliminary Analysis," in *Proc. of the 5th International Workshop on Next Generation of System Assurance Approaches for Safety-Critical Systems (To appear)*, 2016.

[83] L. Getoor and C. Diehl, "Link Mining: A Survey," *SIGKDD Explorations Newsletter*, vol. 7, no. 2, pp. 3–12, 2005.

[84] J. Sowa, "Semantic Networks," in *Encyclopedia of Cognitive Science*. Wiley, 2006.

[85] E. Hatcher and O. Gospodnetic, *Lucene in Action*. Manning Publications, 2004.

[86] H. Asuncion and R. Taylor, "Automated Techniques for Capturing Custom Traceability Links Across Heterogeneous Artifacts," in *Software and Systems Traceability*, J. Cleland-Huang, O. Gotel, and A. Zisman, Eds. Springer, 2012, pp. 129–146.

[87] A. Felfernig, M. Jeran, G. Ninaus, F. Reinfrank, S. Reiterer, and M. Stettinger, "Basic Approaches in Recommendation Systems," in *Recommendation Systems in Software Engineering*, M. Robillard, W. Maalej, R. Walker, and T. Zimmermann, Eds. Springer, 2014, pp. 15–37.

[88] M. Borg, "TuneR: A Framework for Tuning Software Engineering Tools with Hands-On Instructions in R," *Journal of Software: Evolution and Process*, vol. 28, no. 6, pp. 427–459, 2016.

[89] S. Shivaji, E. Whitehead, R. Akella, and S. Kim, "Reducing Features to Improve Code Change-Based Bug Prediction," *IEEE Transactions on Software Engineering*, vol. 39, no. 4, pp. 552–569, 2013.

[90] E. Murphy-Hill and G. Murphy, "Recommendation Delivery," in *Recommendation Systems in Software Engineering*, M. Robillard, W. Maalej, R. Walker, and T. Zimmermann, Eds. Springer, 2014, pp. 223–242.

[91] P. Runeson, M. Höst, A. Rainer, and B. Regnell, *Case Study Research in Software Engineering. Guidelines and Examples*. Wiley, 2012.

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

[93] B. Regnell, R. Berntsson Svensson, and T. Olsson, "Supporting Roadmapping of Quality Requirements," *IEEE Software*, vol. 25, no. 2, pp. 42–47, 2008.

[94] R. Berntsson Svensson, Y. Sprockel, B. Regnell, and S. Brinkkemper, "Setting Quality Targets for Coming Releases with QUPER - An Industrial Case Study," *Requirements Engineering*, vol. 17, no. 4, pp. 283–298, 2012.

[95] R. Berntsson Svensson and B. Regnell, "A Case Study Evaluation of the Guideline-Supported QUPER model," ser. Requirements Engineering: Foundation for Software Quality/Lecture Notes in Computer Science. Springer, pp. 230–246.

[96] S. Panichella, "Supporting Newcomers in Software Development Projects," PhD Thesis, University of Sannio, 2014.

[97] I. Avazpour, T. Pitakrat, L. Grunske, and J. Grundy, "Dimensions and Metrics for Evaluating Recommendation Systems," in *Recommendation Systems in Software Engineering*, M. Robillard, W. Maalej, R. Walker, and T. Zimmermann, Eds. Springer, 2014, pp. 245–273.

[98] 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*, 2005, pp. 135–144.

[99] C. van Rijsbergen, *Information Retrieval*. Butterworth, 1979.

[100] 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*, 2012, pp. 111–120.

[101] K. Spärck Jones, S. Walker, and S. Robertson, "A Probabilistic Model of Information Retrieval: Development and Comparative Experiments," *Information Processing and Management*, vol. 36, no. 6, pp. 779–808, 2000.

[102] 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*, 2011, pp. 275–284.

[103] B. Jansen, "Search Log Analysis: What It Is, What's Been Done, How To Do It," *Library & Information Science Research*, vol. 28, no. 3, pp. 407–432, 2006.

[104] 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*, 2011, pp. 133–142.

[105] 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*, 2011, pp. 1225–1234.

[106] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering: A Practical Guide*. Springer, 2012.

[107] 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*, 2009, pp. 193–202.

[108] B. Schaffer and C. Riordan, "A Review of Cross-Cultural Methodologies for Organizational Research: A Best-Practices Approach," *Organizational Research Methods*, vol. 6, no. 2, pp. 169–215, 2003.

[109] M. Borg, "In Vivo Evaluation of Large-Scale IR-Based Traceability Recovery," in *Proc. of the 15th European Conference on Software Maintenance and Reengineering*, 2011, pp. 365–368.

[110] 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*, 2012, pp. 179–193.

[111] I. Habli, R. Hawkins, and T. Kelly, "Software Safety: Relating Software Assurance and Software Integrity," *International Journal of Critical Computer-Based Systems*, vol. 1, no. 4, pp. 364–383, 2010.

[112] 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*, 2014, pp. 17–24.

[113] 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*, vol. 21, 2010, pp. 1–10.

[114] C. Dantas, L. Murta, and C. Werner, "Mining Change Traces from Versioned UML Repositories," in *Proc. of the Brazilian Symposium of Software Engineering*, 2007, pp. 236–252.

[115] G. Canfora and L. Cerulo, "Impact Analysis by Mining Software and Change Request Repositories," in *Proc. of the 11th International Symposium on Software Metrics*, 2005, pp. 9–29.

[116] 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*, 2010, pp. 119–128.

[117] 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*, 2003, pp. 246–255.

[118] 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*, 2008, pp. 337–345.

[119] A. Dubey and J. Hudepohl, "Towards Global Deployment of Software Engineering Tools," in *Proc. of the 8th International Conference on Global Software Engineering*, 2013, pp. 129–133.

[120] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "MOA: Massive Online Analysis," *Journal of Machine Learning Research*, vol. 11, pp. 1601–1604, 2010.

Markus Borg Dr. Markus Borg is a senior researcher with the Software and Systems Engineering Laboratory, SICS Swedish ICT AB. He received a MSc degree in Computer Science and Engineering (2007) and a PhD degree in Software Engineering (2015), both from Lund University. His research interests are related to alleviating information overload in large-scale software development, with a focus on increasing the level of automation in the inflow of issue reports. Prior to his PhD studies, he worked as a development engineer with ABB in safety-critical software engineering. He is a member of the IEEE.

Krzysztof Wnuk Dr. Krzysztof Wnuk is an assistant professor at the Software Engineering Research Group (SERL), Blekinge Institute of Technology, Sweden. His research interests include market-driven software development, requirements engineering, software product management, decision making in requirements engineering, large-scale software, system and requirements engineering and management and empirical research methods. He is interested in software business, open innovation, and open source software. He works as an expert consultant in software engineering for the Swedish software industry.

Björn Regnell Dr. Björn Regnell is Professor in Software Engineering at the Faculty of Engineering, LTH, Lund University, Sweden. He has contributed to several software engineering research areas including requirements engineering, software quality, software product management and empirical research methods in software engineering. He was ranked among the top 13 scholars in the world in experimental software engineering in IEEE Transactions on Software Engineering, 31(9):733-753 (2005). He is/was a reviewer for several high-impact journals and peer-reviewed conference program committees and he is currently a member of the Editorial Board of the Requirements Engineering journal (Springer) and the Steering Committee Chair of www.refsq.org. Prof. Regnell has published more than 80 peer-reviewed articles in journals and conferences. He has edited several special issues in journals and proceedings and he is co-author of several books including the widely cited "Introduction to Experimentation in Software Engineering" (Springer, 2000) and "Case Study Research in Software Engineering - Guidelines and Examples" (Wiley, 2012). Prof. Regnell worked part time as Senior Researcher at Sony Ericsson, CTO Office, Lund, Sweden 2005-2007, and he works as expert consultant in software engineering for the Swedish software industry.

Per Runeson Dr. Per Runeson is a professor of software engineering at Lund University, Sweden, head of the Department of Computer Science, and the leader of its Software Engineering Research Group (SERG) and the Industrial Excellence Center on Embedded Applications Software Engineering (EASE). His research interests include empirical research on software development and management methods, in particular for verification and validation. He is the principal author of “Case study research in software engineering”, has coauthored “Experimentation in software engineering”, serves on the editorial board of Empirical Software Engineering and Software Testing, Verification and Reliability, and is a member of several program committees.