CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Investigation of approaches for change impact analysis and facilitating finding reviewers in testing frameworks

Master's thesis in Software Engineering

Abel Yohannes Asefa
Fredrik Hansson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY AND UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2018

Master's thesis 2018:NN

Investigation of approaches for change impact analysis and facilitating finding reviewers in testing frameworks

Abel Yohannes Asefa
Fredrik Hansson

Department of Computer Science and Engineering
Division of Software Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2018

Investigation of approaches for change impact analysis and facilitating finding reviewers in testing frameworks
Abel Yohannes Asefa
Fredrik Hansson

Supervisor: Francisco Gomes de Oliveira Neto
Examiner: Regina Hebig

Master's Thesis 2018:NN
Department of Computer Science and Engineering
Software Engineering
Chalmers University of Technology
University of Gothenburg
SE-412 96 Gothenburg
Sweden
Telephone +46 31 772 1000

Department of Computer Science and Engineering
Gothenburg, Sweden 2018

Abstract

As software becomes larger and more complex and time between deliveries decreases the more maintenance becomes an issue. An established way to facilitate some maintenance efforts is to use automated testing frameworks to reduce the time needed for testing, freeing up both computational and human resources for other tasks. However, as the number of testers using such a framework increase so does the maintenance efforts on the framework itself and there seems to be a gap in research regarding processes for improving maintainability in testing frameworks.
This study aims to provide means of improving maintainability of testing frameworks by first investigating issues that hinder maintenance efforts in a real testing framework and search in literature for possible solutions that can be applied to address the issues.
We implement a tool for change impact analysis to assist maintainers in making informed decisions during maintenance and we create guidelines for what to consider when selecting tools for automatic reviewer recommendation.
We evaluate the tool and guidelines by interviewing experienced practitioners and the results show that maintainability of a testing framework can be improved by using tools for CIA and the guidelines provide a good starting point for practitioners wishing to learn about reviewer recommendation tools.
Specifically, we found that by using our tool maintainers can more easily identify the impact of their modifications and understand the related components of the framework, allowing them to mitigate the risk of introducing faults by taking corrective actions. The interviewees pointed out that there are limitations of the tool in the current context, most importantly that a specific type of important files are not covered and that the impact set may sometimes be too large. These are limitations that should be addressed in future research as all interviewees agreed that if they could be addressed, the tool would be very valuable.
Ultimately, our contributions to research are i) fostering transfer of research in CIA to industry and assessing its effect, ii) identifying challenges in implementing tools for CIA in practise and iii) providing means of understanding tools for reviewer recommendation. Thus, the end goal of the thesis goes beyond the deliverables we produce, additionally, we aim to generalize the results of the thesis such that they can be applied in other contexts where organizations using testing frameworks face similar problems. In the long term, our research lays the foundation for continuing transferring research to industry and helping researchers identify where their efforts are most needed in the industry.

Keywords: Maintenance, Testing frameworks, change impact analysis, reviewer recommendation, challenges, coverage.

Acknowledgements

We would like to extented our gratitude and thanks to Francisco Gomes for being our supervisor at the university, helping us in every step of the research. Furthermore, we would like to thank James Harper and Pat Mooney for being our supervisors at Ericsson, helping us get the information needed to accomplish the research and helping us when we struggled. We would also like to thank Ericsson for giving us the opportunity to conduct our research in cooperation with the company. Lastly, we would like to thank everyone who participated in the interviews conducted.

"Thanks be to God for His unspeakable gift!" 2 Corinthians 9:15
I would like to thank Swedish Institute (Si) for funding my studies.

Abel Yohannes Asefa, Gothenburg, Jun 2018

I would like to thank our fellow master students and friends at Chalmers for their support during our work and keepig our spirits high during our struggles.

Fredrik Hansson, Gothenburg, Jun 2018

Contents

List of Figures

List of Tables

1
Introduction

As software becomes larger and more complex and time between deliveries decreases maintenance becomes more of an issue. Research estimates maintenance costs to range between 50-80 percent of total costs in software development projects [38, 41].

There is plenty of research on maintainability in software in many topics such as measuring maintenance effort [18], different activities effect on maintenance and maintenance cost [19, 42], how to improve maintainability from an organizational point of view [20] and software maintenance's effect on cost and schedule. But when it comes to improving the sub-characteristics of maintainability e.g. understandability, how easy it is to understand the code , and modifiability, how easy it is to modify the code without introducing defects, there seem to be a lack of widely known tools and approaches for improving either.

An established way to facilitate some maintenance efforts is to use automated testing frameworks to reduce the time needed for testing, freeing up both computational and human resources for other tasks. However, as the number of testers using such a framework increase so does the maintenance efforts on the framework itself.

Wiklund et al. and Rafi et al. found that maintenance of test systems is an issue and both found that there is little research within the area of automated testing that is based in practise [35, 34]. In certain environments, like that of agile projects, maintenance issues in test systems become even more pronounced. Eldh states that the test suite in an agile continuous integrated build can quickly become costly if there isn't a good architecture or rules in place. In such cases maintainers often add new test cases instead of maintaining existing ones which quickly leads to overlapping test cases and wasted efforts [28].

When it comes to maintenance in testing frameworks there is research in handling test suites with methods and frameworks for reducing size of test suites [21, 91]. However there seems to be a gap regarding processes for improving maintainability in testing frameworks other than reducing the size of a test suite.

Testing frameworks are often at least as complex as their respective System Under Test (SUT) and should be developed and maintained with the same care as the SUT but this is rarely the case in practise [35, 87]. This is likely caused by the high costs associated with doing so. In fact, research has shown that one of the main impediments to adopting automated testing in the first place is the high costs related to

development and maintenance of the testing software [35, 86].

One could say that maintenance issues in testing frameworks are indirect, meaning that while they don't necessarily cause problems in their SUT they allow problems to arise with time as the testing is not done properly. This problem is aggravated by the fact that few companies employ effort in creating test suites for the test framework itself, since that implies the two testing efforts: the SUT and the test framework used [87, 35] as shown in figure 1.1.

Figure 1.1: The two testing efforts

The lack of testing of testing frameworks make defects [84] (also known as faults) harder to detect causing change propagation to become an even larger issue in testing frameworks. Furthermore, issues in testing frameworks are not as noticeable, as they only affect the testing of the SUT, which may function well despite not being tested correctly. Consequently, the number of detected defects in the System Under Test is not a feasible metric to assess whether the test framework is being properly maintained.

For example, a modification of a method in the test framework may affect several test cases using that method. If the modification break a test case, it could still result in the test passing if for example the test case still received/provided a valid input/output from the method, making the error go unnoticed. Even if the test case fails it may not be clear that the test case is broken, and maintainers may spend significant amounts of time trying to fix the SUT without realizing that the defect is in the test system. An example was reported by Karlström who stated that in some situations it was hard to decide whether it was the test system or the SUT that caused tests to fail [88].

Thus, complementary verification approaches to testing (e.g. static analysis) are suitable for this scenario. To our knowledge, there is little or no research on maintainability in testing frameworks or how to improve it.

If the complexity and size of any software (including test frameworks) continues to increase, it is a very real possibility that the currently available methods may not be enough to keep maintenance efforts at manageable levels. Therefore there is value in investigating possible methods for improving maintainability of testing frameworks since it will provide stakeholders with mechanisms to allow evolution of their test frameworks with reduced effort.

One example of the advantage in supporting maintainability of testing frameworks is that doing so allows testers to modify the framework to enable creation of more reusable test cases, by for example adding functions or interfaces to the test framework. We suggest reducing the effort by providing instruments (e.g., data and impact analysis) to inform tester the consequences of those modification on existing test cases. Note that when several testers are simultaneously changing the framework, understanding those changes require more cognitive effort, which is also true when creating new test cases reusing the added features to the framework.

1.1 Problem description

The thesis focuses on two tasks that maintainers perform i) change impact analysis, hereby referred to as CIA, and ii) finding reviewers. To describe the tasks we present a scenario about a maintainer of a testing framework.

> Alice, who started 6 months ago as a new developer at Ericsson, opened her computer and she started reading emails. And one of the emails got her attention:
>
> > "Last Friday we created the new feature for the EPG, so we are hereby asking for new test cases to test the new functionality."
>
> Alice noticed that the issue had top priority and started working on it right away. First, she checked what the new feature was and how it worked and once she understood it, she started writing test cases for it. While doing so she noticed that the `createAll` method in `robustness.py` file (a file in the test framework) has the code she needed. All she has to do is to *add an optional argument to the method*. However, if she changes the `createAll` method to accept an optional argument, she may affect other test cases using the method.
>
> There are few tests for the framework and little documentation so to tackle this problem she started going through the test cases one by one, trying to figure out whether they will be affected. After a couple of hours, she decided that it is impossible to figure out the impact this way and asked for help from her colleagues. However, like her, most of her teammates were inexperienced with the test framework and couldn't provide much help. Feeling frustrated she committed the changes hoping they wouldn't break anything elsewhere.
>
> After committing the changes she had to find a suitable person to review her changes and hopefully help her in determining their impact. However, she doesn't know who is experienced with the part of the framework she made changes for, so she simply added as many people as possible in

the review process hoping that some of them could help in the review.

We list the following risks associated with Alice's modifications. i) Her modifications introduced a defect that may cause the test case to fail. If so, other maintainers may think there is a problem in the SUT directing misguided efforts to finding defects in the SUT. ii) Her modified code may not be invoked when other tests are executed, thus the defect may remain undetected until the `CreateAll` method is executed and even then it may not be clear that the defect in the test method and not the SUT.

When developers perform maintenance tasks they must accurately determine the impact of their modifications to ensure that they don't break the code elsewhere and find competent reviewers to review the code. The reviewers in turn check that the impact was correctly estimated and mitigated. However, both tasks can be challenging to perform, especially if there are no tools available supporting the tasks as Alice experienced.

The two tasks are intertwined in that completing one task may help in completing the other. First, if one is able to identify maintainers who are experienced with the code, one can ask them for assistance in performing CIA. Secondly if one can correctly estimate which parts of code are affected, one can more easily identify someone with knowledge about those parts.

1.2 Research Questions and Contributions

The purpose of the study is to improve maintainability of Ericsson's testing framework. We accomplish this by implementing a tool for CIA with the goal of improving the sub-characteristics of maintainability: understandability and modifiability. Therefore our first two research questions are:

- RQ1: To what degree can modifiability of a testing framework be improved by using a tool for CIA?
- RQ2: To what degree can understandability of a testing framework be improved by using a tool for CIA?

We search in literature for existing solutions for CIA and adapt a tool for CIA by levering from existing instruments at Ericsson. Then, we investigate how to properly use the data collected from the tool to inform testers of maintenance related decisions. Additionally we hope to identify possible challenges in implementing CIA tools in testing frameworks that should be considered. Thus, we include a third research question:

- RQ3: What are the challenges in implementing a CIA tool for a test framework?

In addition to the tool, we hope to indirectly improve maintainability of the framework by addressing the issue of finding reviewers. To facilitate finding reviewer one could either modify the code reviewing process to provide optimal conditions for

finding reviewers or one could adopt tools for performing and/or assisting maintainers in the task. We search in literature for ways to facilitate finding reviewers including best practises and tools. Therefore we also include the following research questions:

- RQ4: What are the current best practices in literature regarding code review and code ownership that facilitate finding reviewers?
- RQ5: What should be considered before adopting tools that support stakeholders in finding appropriate reviewers for their code commits?

We provide a set of guidelines for choosing reviewer recommendation tools and we suggest changes that Ericsson can make in their code reviewing process to facilitate finding reviewers.

The deliverables are a means to lever maintainability at Ericsson test maintenance activities. The end goal of the thesis goes beyond the deliverables we produce, additionally, we aim to generalize the results of the thesis such that they can be applied in other contexts where organizations using testing frameworks face similar problems.

Our research provides the following contributions: i) fostering transfer of research in CIA to industry and assessing its effect, ii) identifying challenges in implementing tools for CIA in practise and iii) providing means of understanding tools for reviewer recommendation.

A recent study shows that most of the contributions in the area of CIA are not transferred to industry, so fostering such transfer and empirically assessing its effect is a contribution to the fields of CIA and maintenance [23].

We find that tools for CIA can be used to improve maintainability of testing frameworks but that there are several challenges that must be overcome for them to become truly useful and adopted in practise. We also find that reviewer recommendation tools can be understood using our guidelines but more research is required to fully capture all important aspects of them.

The rest of the thesis is structured as follows: In section 2 we present the background of the thesis, section 3 presents the methodology, section 4 and 5 presents the results of the interviews, the tool and the results regarding finding reviewers. The results are discussed in section 6 and the different threats to validity and how we have addressed them are presented in section 7. Lastly we present the conclusions.

2
Background

This section describes the background of the thesis including all topics we will discuss. We establish the current state of the art within the related areas of research and justify our research based on the current literature.

2.1 Automated testing

According to Wiklund et al., software testing is the most widely used method of quality assurance in software development [35], which is reflected in its costs which range from 30-80 percent of a projects total costs [35, 37].

Automated testing is a common way to reduce effort required in testing as it increases the efficiency for repetitive steps, especially in regression testing [36]. Rafi et al. found in his research that the main benefits of automated testing are reusability, repeatability and effort saved in test executions [34]. Thus the criticality of testing, its high cost and the possibility to reduce these are the main reasons why it is desirable to automate testing.

However, despite the apparent benefit of automating testing there is surprisingly little empirical research on the topic. Rafi et al. found in their systematic literature review, in 2012, only 25 articles in the area [34]. Five years later in 2017, in a literature review by Wiklund et al. only 39 articles were identified in the area, of which only 18 were experience reports [35].

While Rafi et al. was able to identify the benefits and limits of automated testing and Wiklund et al. was able to identify the current impediments in automated testing, both state that there is a need to increase the body of evidence regarding the actual use of automated testing. Rafi et al. found that maintenance of automated test cases are an issue and that 45 percent of practitioners think that the current tools do not match their needs [34]. The primary themes regarding the test system that Wiklund et al. identify are development and maintenance of tools, tool selection, and configuration of the test beds and test system [35].

According to Wiklind et al., test systems are usually at least as complex as their SUT's and should thus be developed and maintained with the same care as the SUT [35]. However this is usually not the case as test systems in practise are poorly tested, undocumented and unstructured [35, 87]. Under such conditions the main-

tainability decreases as maintainers have insufficient means of understanding the test system. This further aggravates the issues as, for example, maintainers fail to reuse parts they can't understand and instead recreate similar parts that further contribute to a poor test system structure [87].

This lack of support for maintaining, testing and understanding test systems is likely caused by the related costs. The goal of test automation is usually related to saving resources in terms of time and money and if automated testing can't deliver appropriate return on investment it may even be abandoned [35, 87].
Thus it seem that maintenance is a prominent issue in automated testing yet there is not enough research on the topic to provide aid for the industry in addressing it.

2.2 Change impact analysis

Change impact analysis (CIA) is an analysis of what impact a modification has on the rest of the software. When modifying code during maintenance work, it is critical to understand that modifications made may affect other parts than the modified code possibly introducing more defects that need to be fixed, further increasing the maintenance work.

Research on change impact analysis has been done in several areas and techniques have been created for its purpose like the WAVE-CIA technique that first identifies the core (a set of methods affected by the change) and then calculates the ripple effect caused by the change [7]. Research has also produced several kinds of program slicing techniques ranging from static to dynamic that may be used in different contexts to varying degrees [8]. One survey has identified many different approaches to CIA but a majority of these lack tool-support and some are only for specific languages and not applicable elsewhere [1].

All tools for CIA that we found have been evaluated predominatly by quantitative approaches (e.g. comparing algorithms). Some examples of these include Zhang et al. and Ren et al. who evaluated their tools by analyzing their ability to predict which test cases are affected by which modifications [52, 53], Apiwattanapong et al. evaluated their tool by comparing its execution cost and precision to two existing algorithms [3], Orso et al., similarly to Apiwattanpong et al., compared his technique with three others[2], and Zimmermann et al. who evaluated his tool by using it on eight different projects and checked whether the actual changes that were made could be predicted through historical analysis [54].

Despite there being a plethora of CIA tools in literature few seem to be in use in practise. In fact, research suggests that programmers do not use tools developed for CIA and many do not even know the term. It seems developers perform CIA at different stages of development, mostly before a change and immediately after but the practice of CIA is different from the process as described in the literature. This suggests that the industry has not yet benefited from the research on CIA [23]. Thus further research is necessary to either identify new approaches for CIA or modifying

existing ones to make them more applicable to the industry.

In a recent study, a tool for assessing the impact of changes in Simulink models on artifacts like test cases and other models was presented by Rapos, who argued that with increasing use of Simulink models there is an increasing need for tools to manage such models and related artifacts. [80]. His research is very similar to ours but where he works exclusively with Simulink models we work with testing frameworks for large software products. Rapos extended his work by including his tool in a set of evolution tools that can be used to determine impact of changes on a test harness model, suggest changes on it and even generate a new one if necessary [81]. Because of the similarity of our work these extensions may be possible future extensions of our work, extending our tool to providing suggested changes or even making changes if necessary.

2.2.1 Test coverage

Coverage, in the world of testing, is a measurement that shows how much of the code in a SUT is executed as the tests of the system are executed. It is measured by analyzing what parts of the SUT is executed as a test suite runs. It is a common measure for determining the efficacy and adequacy of testing efforts [44].

Research has shown that a higher code coverage brings both a higher failure detection [5, 46, 47, 48] and improves the effectiveness of software reliability estimation [5, 6]. The research on the topic of coverage include improving code coverage [50], optimizing test coverage for the purpose of reducing regression test suite size [49], using coverage to improve fault localization techniques [51] and using code coverage to guide generation of test cases [45]. Most of the research about code coverage seems to be within the world of testing, specifically regression testing.

In this thesis we use test coverage to elicit data which is later used by the CIA tool for estimating the potential impact of modifications in a testing framework.

2.3 Modern code review

Modern, or contemporary, code review is a process in which peers examine software artifacts for defects and alternative solutions. It has evolved from the traditional software inspections to become less rigid and better suit the needs of developer teams [25].

Despite the large body of research on software inspections, there is limited research regarding current best practices in modern code review (MCR). MacLeod et al. states in her research that “the lessons learned in MCR are widely dispersed and poorly summarized by literature. In particular, practitioners wishing to adopt or reflect on a new or existing code review process may find it difficult to know which challenges to expect and which best practices to adopt for their specific development context” [12]. Rigby stated in his research that “Despite a large body of research

on peer review in the software engineering literature, little work focuses on contemporary peer review in software firms." [25]. The research that exists focus on the current best practices and challenges in modern code review [12, 13, 25], how practices have evolved and converged [25], how developers understand changes in code [14] and how one can improve code review through adoption of best practices [12].

Table 2.1 shows the practices that MacLeod et al. suggest for facilitating finding reviewers and addressing certain issues in code reviewing processes that are tied to finding reviewers.

Fist, it may be desirable to improve the response time of reviewers as if reviewers can respond faster they can respond to more reviews, improving the availability of reviewers. Additionally, if the response time is quick the authors will have to wait less before learning whether the selected reviewer was appropriate or if another reviewer should be found.

To improve the response time of reviewers MacLeod et al. only suggests practise R.i [12], which in itself is not very useful. However, if reviewers understand the changes faster it is likely that they can respond faster. Baccelli et al. states that developers could respond better and faster when the author of a review provides context and direction to them in a review [13]. For helping reviewers understand changes and to find relevant documentation MacLeod et al. suggests applying practices A.i, A.ii and A.iii [12], which is also proposed by Bosu et al. in their research [60].

Practise A.iv may indirectly affect the response time of reviewers as it allows the author to identify and deal with low level issues that would just have wasted the reviewers time [12]. This can be taken even further by using more advanced tools, like tools for CIA. Tao et al. found that the third most important piece of information reviewers need, after rationale and consistency, is information about the risk that the modifications breaks the code elsewhere. This is also the second hardest piece of information to acquire [14]. In addition to helping reviewers understanding modifications quicker such information is also beneficial for finding reviewers, because if you know which files are affected it will be easier to find reviewers who has experience working with one or more of the affected files.

To improve quality of feedback Macleod et al. recommend practices R.ii and R.iii [12, 61]. When creating a checklist for code review one could refer to the research by Fu et al., who provides an example of such a list [61], and Bosu et al. who provides a set of characteristics of good code reviews [60].

In addition to the practices already suggested, MacLeod et al. present several organizational practices that may be helpful when adopting new practices and improving the code reviewing process [12]. These are presented in figure 2.2
Practices O.i and O.ii improve the ability to change the reviewing process to address problems that may arise over time. O.iv and O.v ensure that the maintainers

Challenge	Author(A)	Reviewer(R)
Slow response time, unuseful feedback and lack of documentation	i) Submit small incremental changes ii) Cluster related changes iii) Describe and motivate changes iv) Analyze code using tools to fix minor things and provide reviewers with necessary information about risk and other critical pieces of information.	i) Provide feedback as soon as possible ii) Focus on core issues and avoid nitpicking iii) Use or create a review checklist iv) Use code analysis tools to better understand the code and the changes v) Give constructive and respectful feedback vi) Provide reasons for rejecting a change
Finding reviewers	v) Choose reviewers based on experience or need to learn code. vi) Allow self selection if possible vii) Check who to notify other than the reviewer viii) Notify reviewer as early as possible and explain changes	vii) Set dedicated time for reviews viii) Be prepared to iterate and review again

Table 2.1: Challenges and which practices for authors and reviewers address these

Organizational practise (O)
i) Develop,reflect on and revise code reviewing policies
ii) Develop a mechanism to watch for bottlenecks in the process
iii) Build and maintain a positive review culture
iv) Ensure appropriate tools are in place and used
v) Ensure sufficient training is in place for code reviewing activities

Table 2.2: Challenges and organizational practices to improve the code reviewing process

have the necessary tools and knowledge to work according to the reviewing process policies.

There also seems to be a large body of research on the topic of reviewer recommendation and how to use it to improve code reviewing processes [15]. This topic is explained further in section 2.3.1. Thus we have found research in different areas that provide solutions which address the issue of finding code reviewers. Therefore, we lever those existing findings to apply our solutions with respect to reviewer identification.

2.3.1 Reviewer recommendation

Finding competent reviewers in modern code review is challenging as it is hard to determine the expertise and availability of any possible reviewer at a given time [67]. Therefore, significant efforts in research has been given to the area of reviewer recommendation, hereby referred to as RR, and developing tools for the task. These tools attempt to calculate the expertise of available reviewers and provide recommendations for developers looking for a reviewer [67].

RR tools use different kinds of data depending on the approach they use. The most frequently used data include file paths, comments, issue labels, source code and collaboration history. This information was elicited by analyzing tools available in literature and investigating what data the different tools used, thus the list is not exhaustive.

The two most common ways of extracting such data is by looking at either the commit history of candidates which is for example done by the tools of Zanjani et al. and Kagdi et al. [63, 64] or their review history which for example is done by the tools of Zanjani et al. , Hannebauer et al. and Thongtanunam et al. [63, 65, 15]. Whichever you choose to analyze depends on which kind of expertise one wish to determine. Additionally, both can be used to determine several other factors, for example one can find out how often a developer has collaborated with certain reviewers and to what extent by analyzing either.

Systems for pull requests are frequently used for this purpose and the current state of the art around pull requests seems to be geared around solving issues like finding appropriate reviewers, using automated tools for finding reviewers as well as improving these tools [23, 10]. Research has been done on automatic recommendation of reviewers in the context of pull requests [10] and in reviewer recommendation in the context of peer to peer review in software engineering [11].

While there are a large number of RR tools [62, 65, 15, 66, 71, 70, 79, 63, 69, 10, 23, 68] available, we have not found any research that summarizes the current state of the art or provides means of comparing tools. Many tools are evaluated by comparing algorithms [62, 65, 67] to existing ones but there seem to be a lack of benchmarks and means of comparing tools. This makes it hard to identify techniques that suit one's needs.

2.3.2 Code ownership and responsibility

Ensuring that for every piece of code in a software exists a responsible developer that knows and maintains the code is an important goal. Code without responsible developers is a huge risk as that means there are pieces of the software that no one fully understands or maintains.

Code ownership is a term used to describe whether a person is responsible for a piece of code or if the piece of code lacks a responsible developer [72]. It is usually

expressed as who has modification rights over certain pieces of code and is often used as a proxy for responsibility [28]. There is research on code ownership's impact on maintainability [26] and software quality [72] and there is research on different perspectives on code ownership [28].

There are several different models of code ownership each with their benefits and drawbacks. For example, regarding the number of code owners. Having too many people working with a piece of code has been proven to decrease its maintainability [26]. Farago et al. states that the boundaries of responsibility should be as clear as possible to avoid decreasing maintainability [26]. However, you don't want too few code owners either as that may make you too dependent on a small set of people [26, 27]. A motivating example is provided by Bacchelli et al. who states that if the author of a change is the only expert, she has no potential reviewers [13].

Nordberg presents a model for managing code ownership based on the idea of adopting different models of code ownership depending on which phase a project is in, to match the needs of the project with regards to code ownership [27].

He identifies four models of code ownership including:

1. Product specialist - a single individual manages all code with occasional help from others.
2. Subsystem ownership - each subsystem has a specific owner and each team member owns one or more subsystems.
3. Chief architect - one chief programmer has primary ownership of all code but has a team that takes supporting roles in fleshing out the chiefs vision.
4. Collective code ownership - all code is owned by all developers and everyone is free to work across all subsystems and everyone should contribute to all systems.

Nordberg's model states that during the inception of a project one should apply the product specialist model by having one person create the high level features, core requirements and business case as well as articulate the vision of the new or revised system. The product specialist should increasingly accept help from analysts to gather requirements but remains the sole owner. Then as the project grows and the tasks start becoming to large for one individual to manage one should switch to the chief architect model where the product specialist is assisted by a chief architect who focuses on whole system design and utilizes a growing team around him to perform prototyping, planning and setting up environments with the goal of establishing a clear architecture [27].

Once the architecture is well-defined one should start switching over to collective code ownership as the number of developers increases to fit the needs of the project, allowing rapid propagation of the architecture to a growing team of developers. During the switch to collective ownership, the chief architects should be moved to new products, which could be subsystems within the same project, to repeat the same cycle. Finally as subsystems of the project are becoming stable one should switch to

the subsystem ownership model by assigning individuals or smaller teams to finalize the subsystems.

Current research states that the definition of code ownership as having modification rights causes problems [28]. Thongtanunam et al. states in their research that approximations of code ownership should take review activities into account as well as he found that many developers who contribute little in terms of code may actually be suitable code owners if they often perform reviews of the code. In fact he found that many developers only contribute to modules through reviewing code changes for them [16]. Therefore one should consider modifying the definition of ownership before using it for managing responsibility. Owning code should not mean having modification rights but rather being responsible for understanding, maintaining and reviewing a piece of code.

3
Methods

We performed a case study at Ericsson AB with the purpose of improving maintainability of their testing framework. We accomplish this by implementing a tool for CIA with the purpose of improving understandability and modifiability of the framework. Additionally, to further foster transfer of research on CIA to industry we identify and discuss challenges we encounter when implementing a tool for CIA. Therefore our first three research questions are:

- RQ1: To what degree can modifiability of a testing framework be improved by using a tool for CIA?
- RQ2: To what degree can understandability of a testing framework be improved by using a tool for CIA?
- RQ3: What are the challenges in implementing a tool for CIA in a test framework?

By answering how well the tool helps in improving these sub-characteristics we can then attempt to quantify how much the tool helps in improving maintainability.

To answer the first two research questions we define two main objectives of the tool: i) Helping maintainers understand the test cases, the TF and dependencies between them and ii) helping maintainers understand the potential impact their modifications may have so that corrective action may be taken to avoid introducing defects. These objectives will be the main focus of the evaluation of the tool, which is described in detail in section 3.5.

To answer RQ3 we will take note of any significant challenges we encounter during the implementation process and at the end of the thesis conduct a retrospective analysis to summarize which challenges we have faced, how we have overcome them and what the implications are if the challenges are left undressed.

In addition to the tool, we hope to indirectly improve maintainability of the TF by addressing the issue of finding reviewer, thereby making the maintenance process more efficient. We accomplish this by investigating both practices for improving code review processes and tools for reviewer recommendation. We chose not to implement a tool for facilitating finding reviewers because Ericsson could not state what kind of solutions they were interested in nor express the requirements of such solutions. They stated a need for understanding the options available and thus we had to perform an exploratory study on the subject.

Therefore our research questions regarding finding reviewers are:

- RQ4 What are the current best practices in literature regarding code review and code ownership that facilitate finding reviewers?
- RQ5 What should be considered before adopting tools that support stakeholders [1] in finding appropriate reviewers for their code commits?

A summary of our case study planning is presented in table 3.1

Objective	Improving and exploratory
The context	Maintainability of test frameworks
The case	Test framework at Ericsson
Theory	Change impact analysis, reviewer recommendation, code ownership
Research questions	RQ1, RQ2, RQ3, RQ4 and RQ5
Methods	Interviews, instrumentation and documentation analysis

Table 3.1: Our case study planning according to guidelines presented by Runeson and Höst [40]

3.1 Our approach

We elicited information about the current working process by interviewing users of the test framework to identify problem areas that gave rise to the issues and to prioritize them according to their criticality. The interviewees were six experienced users with the framework, from different teams and roles which are explained further in section 3.2.2. The interview instrument used in the initial interviews is found in Appendix A

Additionally, to understand the structure and the components of the framework as well as how developers handle it, we reviewed its code and documentation as well as the interview data collected from the developers and testers as shown in figure 3.1.

To find approaches addressing the issues, we elicited information regarding the current trends in Software engineering by analyzing how the issues are addressed in literature, like the IEEE Standard for Software Reviews and Audits [82] or IEEE Standards for Software Maintenance [83].

The purposes of the investigation was i) to lay the foundation for developing a tool for one of the issues that will help humans in making decisions during development and ii) to find ways one can facilitate finding reviewers.

[1] The stakeholders are any maintainers of the TF that needs to find reviewers in their work including testers and developers

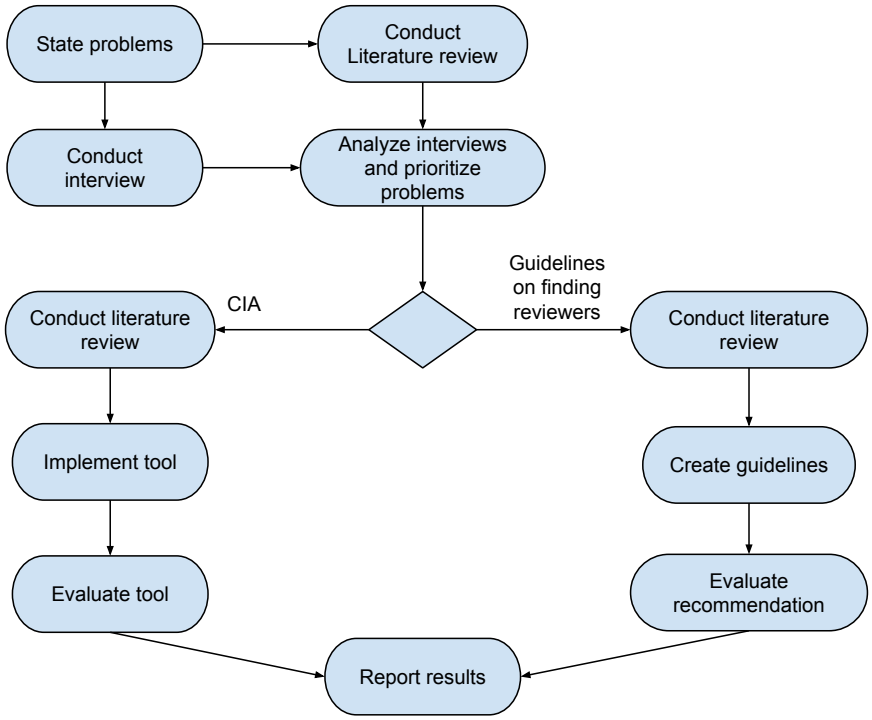

Figure 3.1: methodology activity diagram

3.1.1 Process for reviewing literature

We conducted a literature review in preparation for the thesis and two reviews during the thesis work: i) one mainly focused on CIA, and ii) one focused on finding solutions for how to facilitate finding reviewers. We used the same methodology during all literature reviews, which is described in figure 3.2.

First, we identified general topics of research that may be of interest and then for each area we identified keywords with which to search for articles, for example software testing, test execution, change impact analysis and code reviewer. We analyzed the titles of articles we found to determine if they were within the topics of interest in which case we would save them for closer inspection. Next we read the abstract of all these to determine if they were within the scope and if they were, we would continue to read them thoroughly.

For every article we read that we deemed was within our scope, we attempted to use a snowballing approach in two ways. First, to find new keywords with which we could perform further searches and second to find new sources by looking at their references. Thus our approach has been highly iterative.

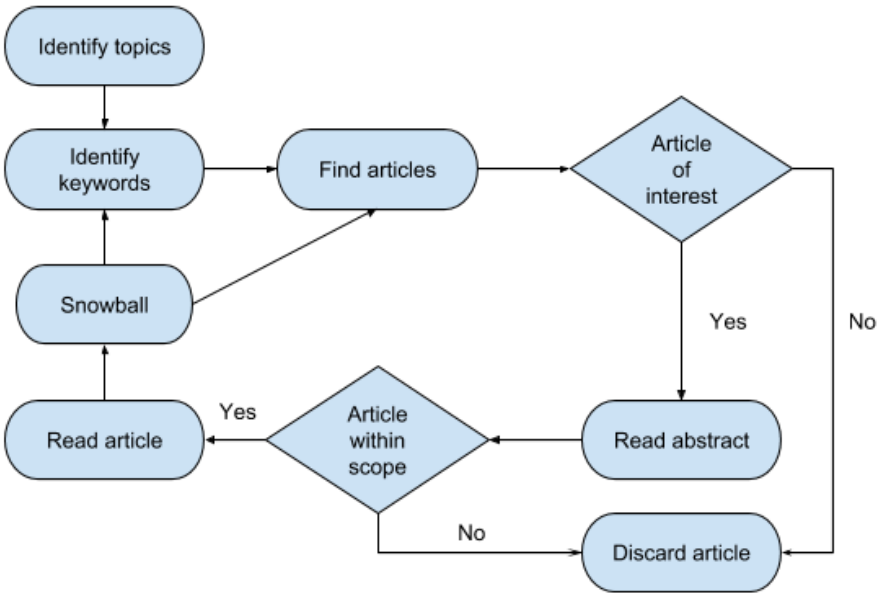

Figure 3.2: Our process during literature review

3.2 Case company

In this section we introduce the case company, Ericsson, by explaining the testing framework they use, their code reviewing process and the challenges that exist in the process.

Ericsson is one of the leading providers of Information and Communication Technology (ICT) to service providers with more than 97,000 employees world wide. The study was conducted at Ericsson located at Lindholmen, Gothenburg. Ericsson provided the study with access to the testing framework, documentation, training material and knowledgable personal for aid in working with the framework and for interviews.

3.2.1 Testing framework

The Ericsson product in question is regression tested continuously especially at weekends since hardware resources are expensive and there is always a very high utilization of these resources during weekdays. To automate the testing effort a testing framework is used and it has many users (usually more than 150 testers), and everyone has access rights to make changes in the framework. The framework itself consists of several parts as shown in figure 3.3.

There is a set of system testers that maintain the test cases and the framework. When a test is run, an automation tool called AutoTT uses a test case, which is fetched from the test case database TCDB, to configure how the test is going to be run and then runs test scripts that execute the test. These in turn use parts of LIBgeneral, which is a repository of libraries, in their execution. Furthermore there

are hooks, test scripts which test special/certain cases, that TCDB sometimes uses. This thesis will focus on the part of the test framework that is inside the grey box as the issues are most prevalent in these parts. We will refer to this part as TF for simplicity's sake.

Since a large part of the code may not be ran until the weekend, failures in it can result in missing or incomplete test status. Coupled with the feedback time for those changing this code it can be very expensive if faults in this code are not found early. One such example could be that one defect could set back project goals with one week since re-execution is needed during the week which causes delays for others utilizing the same resources.

Thus, increasing the maintainability of the TF by increasing understandability and modifiability of the TF may significantly reduce maintenance costs.

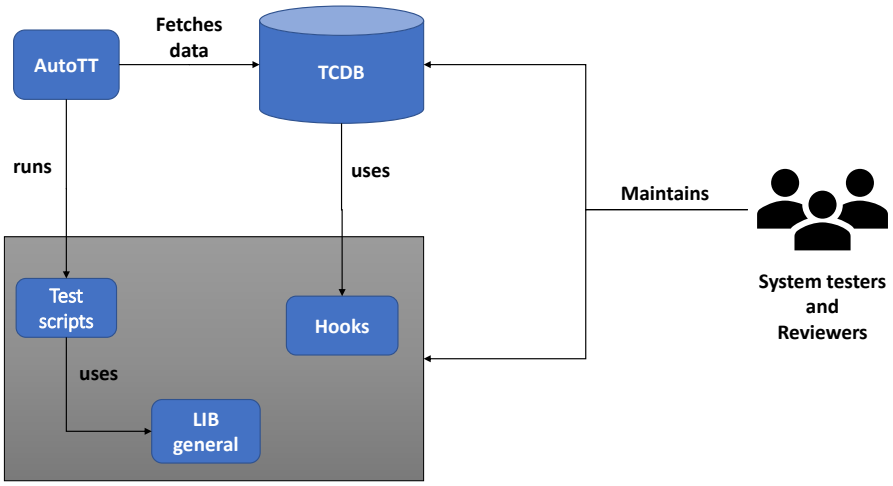

Figure 3.3: The test framework

The two main issues in the TF that need to be addressed to improve its maintainability are: i) estimating impact of modifications and ii) finding reviewers.

When modifying the TF, there is a significant risk that a large number of test cases are affected. This is very hard to detect as there is no current way of seeing what impact any modifications may have and these may cause cascading changes in the system. Thus whenever a developer modifies the TF, there is a significant risk that a set of test cases will fail, not because of the system under test itself but because they are using some part of the TF that has been modified. Therefore, CIA in our context is analysis of impacted test cases caused by modifications in the TF.

Furthermore, when a maintainer modifies a part of the TF, there is, currently, no

instrumented support at Ericsson to know who should review their changes. Due to the difficulties in finding ideal reviewers, it is difficult and risky for an arbitrary tester to accept modifications and guarantee that their new modifications are valid.

These issues are currently a problem in Ericsson's process and in the future the number of test cases and working developers may increase, further accentuating the issues. Therefore the current process must be changed to address these issues.

3.2.2 Code reviewing practices for test code at Ericsson

There are three main roles among the maintainers of the framework including XFT, LSV and TOR. XFT's (cross functional team) are responsible for adding new features to the product and also new testcases or updates to testcases to test the new feature. LSV's (Latest system version) are responsible for the regression testing. TOR (Test object responsible) are four individuals each responsible for a specific test objects, which include stability, robustness, capacity and IRT (installation and replacement test).

When an XFT or LSV has made changes/added functionality they send their code for review to the corresponding TOR. The TORs review and/or assign reviews to other people from XFT or LSV depending on who would be best suited to perform the review.

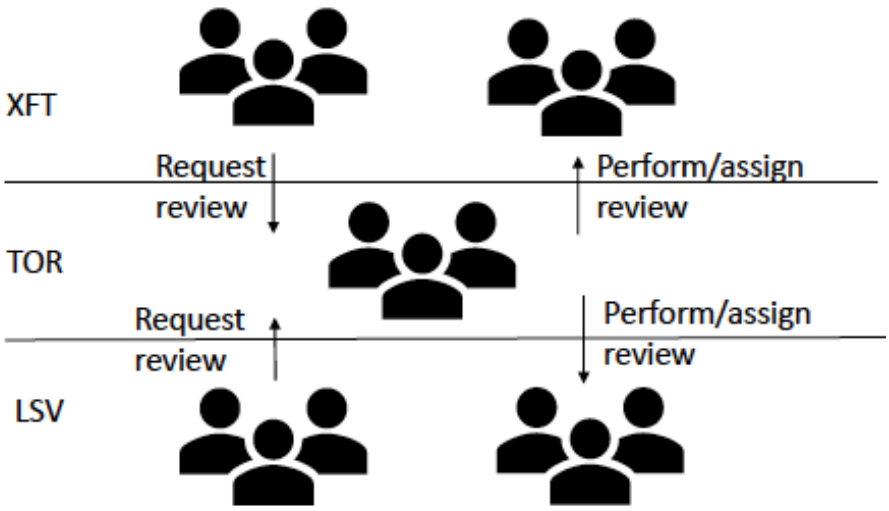

Figure 3.4: The different roles in the TF

Any code that is produced at Ericsson must be reviewed before it can be accepted and every modification should be reviewed by at least two reviewers and any major modification must also be approved by the areas TOR. All XFT's and LSV's working with the product know who each TOR is and which areas they are responsible for.

To ensure that all code is reviewed, Ericsson uses a code reviewing tool that allows users to submit modifications for review and selecting reviewers as shown in fig-

ure 3.5. The tool enables reviewers and authors to discuss issues and saves all such information for traceability purposes, including reviewers and authors of issues, comments, projects, branches, files, commit histories and more. For a developer, you can see all issues she has worked on or is working on as well as any issue she has reviewed.

However, the tool does not explicitly forbid a single reviewer from giving a high enough score to pass, nor does it explicitly require certain people, for example TOR's, to review it. Thus, it is possible to violate the practices described above. This is one reason why traceability is necessary because if one can trace all issues, any such violations can be found and the committed changes can be removed.

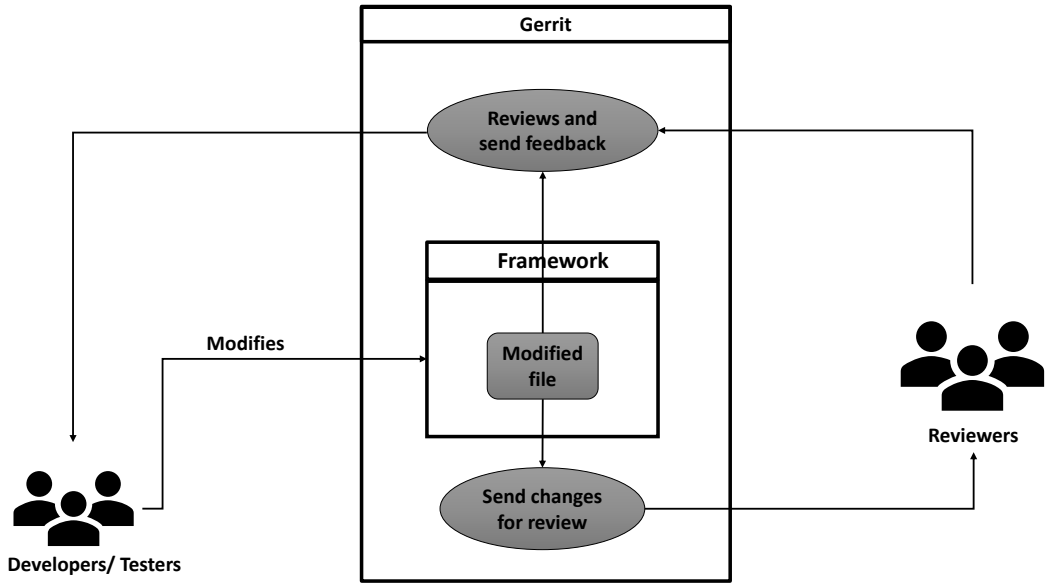

Figure 3.5: The working process of developers, testers and reviewers using Gerrit

Ericsson applies continuous integration. All changes that are submitted are kept at small sizes so reviewers can easily understand the changes with less explanation required from the author, which helps reducing the time required to perform reviews. Furthermore authors generally describe their changes, sometimes providing motivations for why the changes are being made.

Teams at Ericsson are encouraged to create and run unit tests for all code they produce to ensure that any code that is submitted for review has already been tested. This practise is rigorously followed by all teams working with the SUT, but it is not a strict requirement for teams working with the TF and so most such teams *do not follow* this practise.

3.3 Change impact analysis: Finding a suitable technique

In order to identify what CIA techniques will suit our needs, we used Li et al.'s framework for comparison of CIA techniques [1]. The framework presents a set of seven properties used to characterize CIA methods so that they can be compared. Our results in using this framework for finding a technique for CIA are presented in section 4.2.

The properties of the framework are:

1. Object - The change set and the source of analysis
2. Impact set - The output of the analysis composed of the impacted elements
3. Type of analysis - The type of analysis that is performed. The existing types of analysis are static and dynamic analysis. Each has some sub-types as shown below:
   (a) Static Structural - analysis of structural dependencies of the code.
   (b) Static Historical - analysis of change history repositories.
   (c) Static Textual - analysis of comments and or identifiers in the code to identify conceptual dependencies (coupling).
   (d) Dynamic Online - the analysis is done as the program executes and while data is being collected.
   (e) Dynamic Offline - the analysis is done post execution after all data has been collected
4. Intermediate representation - The representation of the source code used by the tool.
5. Language support - What programming languages support the techniques.
6. Tool support - If there are tools supporting the techniques.
7. Empirical evaluation - whether or not the techniques have been empirically validated.

3.4 Process for finding ways of facilitating reviewer identification

While it is possible that the issue of finding reviewers may be addressed by using a tool for reviewer recommendation, it is also possible that the issue can be addressed by dealing with the root causes of the problem. Therefore we elicited information about the working process of developers in the first set of interviews to find such root causes. Specifically we asked how developers at Ericsson find reviewers for their code and what challenges they face. The findings are presented in section 4.1.

After identifying the root causes we decided to investigate three main topics in literature: i) code review best practices, ii) code ownership and iii) tools for automatic

reviewer recommendation.

First we investigated the current best practices in code review including, known issues and practices that address them. The purpose was to identify practices, not already in use at Ericsson, that may address the root causes of the issues in the current code review process.

One of the main root causes identified was that there was no responsibility of files in the TF, meaning that there are no developers responsible for understanding and maintaining the files. Therefore we investigated the concept of *code ownership* to find solutions for how to handle responsibility in the TF. Specifically we tried to find models or approaches for managing code ownership that could be applied to create and maintain a clear structure of responsibility in the TF.

Lastly we investigated tools for RR to find out what tools were available and what different techniques are used in these. Due to the uncertainties in what the requirements would be for such a system at Ericsson we decided to investigate what factors one should consider before implementing such a tool with the goal of providing Ericsson enough information to be able to take the first steps in adopting RR tools. We did this by analyzing different tools to identify common factors among them that one may need to consider, for example, if there are common ways of gathering data.

Based on our findings in literature we compiled a recommendation for how to change the current code review process by adopting best practices, a recommendation for how to manage responsibility of code by applying a modified model for managing code ownership and lastly a set of guidelines for what to consider before adopting a tool for RR.

3.5 Evaluation methodology

In this section we present the method used to evaluate the tool for CIA and the guidelines for choosing a tool for finding reviewers. First we describe the objectives of the evaluation, then we describe the plan, the process and finally the interview instrument and our plan for validating it.

3.5.1 Objectives of the evaluation

As described at the start of section 3 the two objectives for our tool were i) to help maintainers understand the test cases, the TF and dependencies between them and ii) helping maintainers understand the potential impact their modifications may have so that corrective action may be taken to avoid introducing defects.

Therefore the objective of the evaluation was to determine how well the tool achieves these objectives. This in turn allows us to determine the usefulness of the implemented tool for CIA, in terms of increasing modifiability and understandability of

the TF thereby answering RQ1 and RQ2.

As for the guidelines we had to determine if they were useful for understanding and selecting RR tools that suits one's needs and if there were relevant factors not included in the guidelines. By determining this we can verify whether or not the factors we included in the guidelines are relevant as well as identify missing factors allowing us to complement and validate our results regarding RQ5. Therefore we include this as the last objective of the evaluation.

3.5.2 Evaluation plan and process

The target population were developers and testers at Ericsson that are currently working with the TF that we implemented the tool for. Thus, the population were small and their availability was limited due to their workload. Therefore the interviewees were selected by our industry supervisors as they knew who was available and who would be working on the parts of the TF we implemented the tool for. Five maintainers were interviewed in the evaluation, including two TOR's, one XFT's and two LSV. The evaluations were ongoing for the last three weeks of April (9th-30th) and during this time we had maintainers try using the tool and then we interviewed them about their experiences.

Through the interviews, we elicited information about the tool's usefulness, the working process of the maintainers as well as improvement suggestions for the tool itself. The improvements will be incorporated during the adoption of the tool to the maintenance toolkit, after the conclusion of the thesis work.

To evaluate the usefulness of the guidelines for choosing RR tools we applied the them in the context of Ericsson to identify suitable tools and presented the results along with the guidelines to a TOR. The TOR was then interviewed to provide feedback regarding the usefulness of the guidelines and the results. The TOR was selected for the evaluation because of their extensive experience within the review process and their good knowledge of the testing framework and Ericsson.

The evaluation of the CIA tool used the following process.

Training: The participants were given the implemented tool and we explained its purpose. To explain the usage of the tool we provided them with use cases for the intended purpose of the tool and gave them access to a wiki explaining the tool.

Usage: The users then returned to their daily work and tried using the tool in their work.

Interview: After testing the tool each user was interviewed about their experiences using the tool and asked about its usefulness. Each interview was recorded for later transcription and analysis.

Analysis: In order to save time and be able to perform as many evaluations as possible we began transcription and analysis of elicited data from each interview after they had been conducted and in parallel to having other users trying the tool.

We analyzed the transcribed interviews qualitatively by identifying recurrent themes.

The evaluation of the guidelines for selecting RR-tools did not include any training, and the usage was limited to a presentation of the guidelines and the recommended application of them.

3.5.3 Interview instrument

The interviews were semi structured, as described by Runeson [39], where we asked both open and closed questions. We used semi structured interviews because we needed to catch a possibly wide array of topics. Each user may have different experiences and to fully capture this we had to be able to adapt the interviews by asking follow-up questions and possibly deviating from the order of the questions. However, we had very clear objectives that we needed to answer so we could not use completely open interviews either, as we need a specific set of questions answered.

Therefore we designed our interviews to follow the Funnel shape described by Runeson [39], where we started by asking more general and open questions in an attempt to capture as much knowledge as possible without introducing bias from possibly leading questions. We continued by narrowing the discussion down to the subjects of interest by asking more direct questions aimed towards the subjects of interest for our research.

The interview instruments for the CIA tool and the guidelines are both found in appendix A.

The constructs we use should comply with the existing constructs from state of the art CIA research, so that we are using the right tool. We used the guidelines by Pfleeger and Kitchenham [73, 74, 75, 76, 77, 78] to evaluate the validity of our research in addition to Runeson's guidelines [39] to validate the interview instrument.

To ensure construct and criterion validity of our interview instrument we sought to identify similar instruments used in previous research. However, as mentioned in section 2.2, not one of the tools we investigated, which other than those previously mentioned include tools by Acharya et al., Breech et al., Poshyvanyk et al., Gethers et al., Buckner et al., Gwizdala et al. and Li et al. [8, 4, 55, 56, 57, 58, 7], were evaluated by interviewing users that were allowed to test the tool. This finding is no surprise given that Jiang found that few, if any, instruments and tools used in CIA are in fact transferred to industry [23]. The fact we have not been able to find any tool that has been evaluated by actual users may be a contributing factor to the slow adoption of the many techniques for CIA.

This lack of instruments and sources to use for comparison when designing our own instruments is an inherent threat to the validity of our instruments.
However, we rely on the sessions with experienced practitioners to validate the content of our constructs and their value to Ericsson's maintenance processes. Additionally, we reviewed the interview instrument in different sessions with our academic supervisor to check for possible sources of bias and refined the questions.
To further mitigate this threat, we analyzed interview instruments used in research in other areas to assist in creating our own [12, 89, 90]. Specifically we analyzed how the researches established a purpose with each questions and how they tied their questions to their objectives.

Certainly, we hinder our external validity, since generalization is limited given the current construction and content validity. Conversely, our methodology contributes to the field, by including reusable research instruments for future work involving technology transfer of CIA techniques and tools.

4
Results CIA

In this section we present the results of the main part of the study. First the results of the initial set of interviews are presented and then we present the tool that was developed. We describe the requirements of the tool which were elicited using Li et al.'s framework [1], the implementation of the tool, its process and the results of its evaluation.

4.1 Results of interviews

The interviews provided us with the following insights regarding the working process at Ericsson:

1. CIA
   (a) There are no tools that are used for performing CIA.
   (b) Some teams working with the TF use unit tests to assess the impact of changes but it is not common practise.
   (c) There is no general process for how to do CIA.
   (d) Many rely on their experience to determine the impact of changes.
   (e) The file structure is unclear.
   (f) Figuring out whether modifications break the code elsewhere is very important.
2. Finding reviewers
   (a) There are no tools for reviewer recommendation
   (b) Many stated that finding reviewers is a problem because people outside their team, excluding the TOR, don't have the necessary knowledge about their code to give good reviews.
   (c) Some of the interviewees stated that the response time of the reviews is too long.
   (d) There is no responsibility of files, meaning there are no people assigned to be responsible for files.
   (e) There is a significant lack of documentation for code, except for some code that was developed recently
   (f) The system testers have limited knowledge about areas outside their expertise.

These findings show that there are several challenges in the current working process. The lack of documentation and tools for analysis of TF code at Ericsson make it difficult to estimate risk or trying to understand a piece of code as developers have

to rely on manual approaches for these tasks, often using debugging tools.

Another issue that slows down the process is that there is no standard for structuring files and as a result there are many paths in the TF that contain unrelated files which may affect maintainers ability to navigate and understand the code. We believe that the root causes of why it is hard to find reviewers are related to findings 2.a, 2.d, 2.e, 2.f.

Since the TOR's are few they can quickly become overburdened by reviews and currently the response time of the TOR's is an issue as it may take several days before a response is given, according to several interviewees, which is likely caused by the high workload on the TOR's.

While we can't determine if finding 2.b actually reflects reality at Ericsson it indicates that there is either a problem in sharing of knowledge between teams or that there are not enough means for users to determine the competence of other developers.

Some also mentioned that the feedback they get is sometimes not useful, focusing on minor issues while overlooking more serious ones. If a TOR is performing the review, they may not have time to review the code in enough detail to give useful feedback. On the contrary one interviewee stated that because of the TOR's wide knowledge of the code, the feedback the TOR's provide may be incomprehensible for the author as it requires a wider understanding of the code than the author possesses.

If someone other than a TOR is performing a review, the quality of the feedback will be heavily impacted by their previous knowledge of the code and their ability to gain such knowledge. The lack of documentation slows down this process as reviewers have less information available for understanding the code, which likely affects the usefulness of the feedback they can provide. Furthermore there are no established guidelines for how to write reviews or how to perform CIA so depending on who performs the tasks, the process may differ.

4.2 The tool for CIA

To find a technique that suited our needs we first identified the requirements imposed by the context at Ericsson and elicit what they meant in terms of the properties in Li et al.'s framework [1], which we presented in section 3.3. We then used the framework to identify what methods or approaches are available for implementation. We describe this process in detail in the following sections.

4.2.1 Object of analysis:

The object of analysis refers to the *change set* and the *source*. The change set is the set of modifications made that one wishes to determine the impact of. The change set can have five different levels of granularity depending on the need of the analysis

including file, class, method, field and statement changes. A change set can also include textual changes.

The source refers to the code that is analyzed and it can either be a single version of the source code or multiple versions. The source may also use execution data, which is usually used to improve the precision of the impact set.

In our case the granularity of the change set would likely need to be on the code statement level because our goal is to find out what parts of TF are affecting what test cases. Having a finer granularity will greatly assist in providing a clearer picture of the real impact as one is able to check the impact of just the modified code statements.

The requirements regarding the source of analysis in our case are less clear as they greatly depends on the type of analysis we choose. If we were to choose historical or textual analysis it is likely that we could run analysis on multiple versions of the software as we would analyze the history from the version control system rather than the system itself.

One other option would be to rely on execution data for the analysis by for example using coverage information. Using coverage information we could perform CIA by mapping the dependencies between different test cases and the parts of the TF they use. Furthermore, by analyzing coverage we can also identify how frequently parts are being used, which could be helpful in determining the risk of modifications. Thus using execution data may be a good option as it could directly assist in achieving our goals with the tool. However, using execution data puts more demanding requirements on the object of analysis.

To gather execution data the regression suite must be executed and since it is so large it can only be run *once a week*. Therefore we can't gather data from multiple versions at the same time as that requires the entire test suite to be run several times with different versions and there aren't enough resources to do so.

Even though it would be possible to save data from weekly executions (e.g., to be used in later analysis), older information may become obsolete because of the high frequency of change, and the inclusion of such data would lower the accuracy of the results. Therefore it would be best to only use data from the current version during analysis.

In summary, if we were to use a method relying on historical or textual analysis we could use multiple versions of the object of analysis. But if we were to rely on execution data in the analysis we would have to use a single version of the object.

4.2.2 Impact set

There were two main goals with the CIA tool we implemented. First it needed to help maintainers in understanding the test cases, the TF and the dependecies between them. Secondly it needs to help maintainers in understanding the impact that any modifications they make in the TF may have on the test cases. The impact set of any chosen tool would have to provide the necessary information for these goals.

In Li's framework, the levels of granularity for the impact set are the same as for the object. To see which parts of the TF were used, we were only in need of an impact set on the file level, showing which test case files use the selected file or code statement.

To achieve our goals we do not need a finer granularity than file level in the impact set as we are merely interested in which test cases are using which parts of the TF. The test cases themselves are not directly using the parts of the TF, rather they instruct the automation tools which then uses scripts that run the TF. Thus, there is little interest in examining the test cases on a finer granularity. Furthermore, choosing a granularity of file level helps keeping the size of the impact set on a manageable level.

4.2.3 Type of analysis

Regarding the type of analysis, the choices available were very limited because of the context. Because there are no explicit dependencies between the TF and the test cases in the code, static structural analysis can't be used to elicit information about such dependencies, as it relies on explicit dependencies in the code to perform its analysis. While textual and/or historical analysis may provide insights of conceptual dependencies, it is likely that the impact set generated would become too large to handle and be too inaccurate [2, 3]. Because our aim is to create a tool that can be used whenever making changes, the information has to be more precise and easy to handle. Therefore dynamic analysis was a more appropriate option as it, while being computationally costlier than static analysis, is more precise [1, 4].

In fact, dynamic analysis has some specific benefits in our context in that it may help in removing dead code[1] and its accuracy is not impacted by the presence of dead code as dynamic analysis relies on execution data. A finding from the interviews was that there is a lot of dead code in the TF. If we were to use static historical or static textual analysis, it is likely that dead code would be included in the analysis which could affect their accuracy. Thus this finding further strengthens the choice of a dynamic analysis technique.

A potential problem with dynamic analysis is that it tends to include more false negatives, not showing parts that are affected by modifications, than static analysis [1, 4]. However, Breech also states that when the safety of the impact set is not

[1]Dead code refers to code that is either never executed or whose results are never used.

critical for its use, they may be more practical to use than sets generated from static techniques. In our context, the safety is not critical as the goal is to aid developers in performing CIA and maintaining a testing framework rather than it's SUT. The worst case scenario an overlooked false negative could have, meaning that the tool missed some dependencies, is that a set of test cases unexpectedly fails when a part of the TF is modified despite the developers having taken actions to ensure no test cases should fail. Furthermore, many false negatives would likely stem from test cases that are not being run, which would be test cases that are not part of the regression suite. These could be interesting to investigate to determine if they can be removed or not but are not critical because of their exclusion from the regression suite.

Regarding the type of dynamic analysis, we could only choose one because of limitations of the context. Online dynamic analysis is not desirable because as we have already stated in section 4.2.1, if we were to rely on execution data, we are limited to gathering such data once a week. Therefore the best choice for type of analysis is dynamic offline analysis. Note that dynamic offline analysis does not hinder us from gathering data during the week if specific test cases are modified and executed, it only means that the analysis of the elicited data does not have to occur as we are eliciting it.

Because we need to use dynamic analysis we can finally determine the requirements regarding the object of analysis. We have to rely on a single version of the object.

4.2.4 Intermediate representation

We were unable to identify any specific requirements regarding the intermediate representation of a tool for CIA. The representation must be accurate and be able to represent the source code without loss of information. Because we could not identify any restricting requirements, we did not consider this factor further.

4.2.5 Language and tool support

The TF is written in python. Thus, we needed a tool that is available for python. The tools available for any language or specifically for python include:

1. Jimpa - Eclipse plugin that analyzes a change request description and a set of historical source file revisions. Identifies impacted files by referencing similar past change requests [59].
2. IRC2M - Computes conceptual coupling which can then be used for impact analysis [55].
3. LDA - Computes relational topic based coupling which can then be used for impact analysis[56].
4. Rose - Eclipse plugin that analyzes version history and a change set to determine likelihood that further changes should be applied in a given section [54].

All four of the available tools are either using historical or textual analysis and so it seemed there were no tools for CIA suitable for our requirements.

4.2.6 Empirical evaluation

Empirical evaluation refers to whether or not techniques have been empirically validated. Any technique for CIA should be empirically validated so that different techniques can be compared to each other [1].

The requirements regarding empirical evaluation in our case are simple. We prioritize techniques that have been evaluated quantitatively and/or by actual users as then we can compare the accuracy against other techniques and gain information about what users need from such tools.

4.2.7 Choice of technique

As shown in figure 4.1, an ideal tool for our requirements would be a dynamic offline tools that analyses data from one version of the object on the code statement level, identifies impacted test cases on the file level showing for a selected part of the object what test cases are affected by it.

We need a tool available for Python and in Python there is a module, called *coverage*, which dynamically collects coverage information that could be used for CIA. Therefore we chose to use that module to elicit coverage on the TF as the test cases are executed during the weekend test suite runs. The elicited information is later analyzed to identify what parts of the TF are used by what test cases.

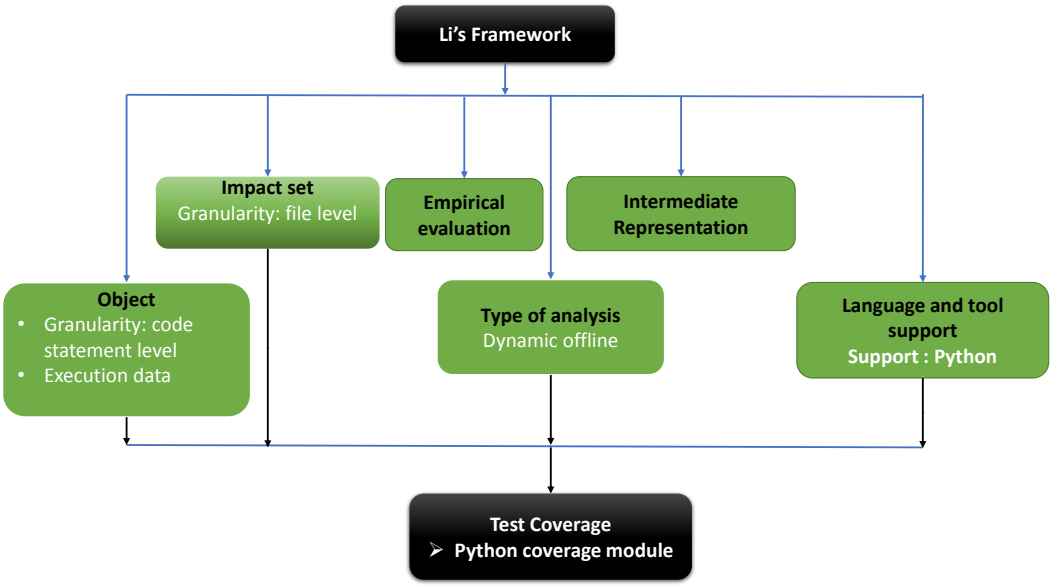

Figure 4.1: Ericsson's context according to Li's Framework

Coverage in testing frameworks has some unique aspects to it compared to other software. Usually, for any piece of software, software coverage tools shows what parts of the code is tested or not, which assumes the existence of a set of test cases for the investigated software.

Conversely, we ran coverage on the TF instead. In other words, we reversed the approach, by using the existing test cases to indicate which parts of the TF were being used or not. Note that the test cases were not for the TF, rather they were test cases used on the SUT.

In contrast to a regular SUT, having parts in a testing framework that are not covered isn't necessarily a bad thing since it only means that those parts are not being used in the creation or execution of tests, rather than not being tested. This can for example mean that there are features that have not been discovered by its users (e.g. testers at Ericsson) or that there is dead code in the TF.

Thus, by gathering coverage data on the TF when executing test cases we aim to identify what parts of the TF were used by the test cases and to what extent. For each test case of interest we can then check what parts of the TF were executed by it and for each part of the TF we can find out what test cases it affects.

4.3 Implementation process

The implementation of this tool was broken down in several steps. First, we elicited coverage information by modifying AutoTT such that the module *coverage* is run on the TF whenever a new test case is executed on the SUT. Then the resulting reports generated by *coverage* are stored in traceable locations, hence connecting reports to their corresponding test cases.

Secondly we created scripts for performing CIA on the TF on a file level by searching through the elicited information based on a selected file in the TF. Thus the impact set will be generated when the user selects a part of the TF and enters it into the tool. The impact set is created by using scripts that search trough the set of coverage reports that were created during the coverage elicitation.

Then we had to increase the fineness of granularity, regarding parts in the TF, to the code statement level showing what code statements in the TF are affected by what test cases. This requires more complex scripts that for every coverage report examines every part of the report.

Finally, because we were developing a new tool we had to empirically evaluate its usefulness. We describe this further in section 3.5

4.3.1 The purpose and process of our tool

As mentioned in section 3 the two main objectives of the tool are: i) To help maintainers in understanding the test cases, the TF and the dependencies between them and ii) to help maintainers in understanding the impact that any modifications they make in the TF may have on the test cases. The tool accomplishes this by providing information about what components in the TF are used by what test cases and vice versa, so that maintainers can determine which test cases are impacted by modifications in the TF.

The process of the tools is shown in figure 4.2. When the regression suite that tests the SUT is run over the weekend, the tool runs a coverage analysis over the TF for every executed test and stores the data in the log files of each specific test case. At the end of the regression suite's execution, the coverage data is parsed and stored in a file as a hash table[2]. The build name is used as an identifier for each weekly run and each identifier is unique.

Then when a developer wants to modify a part of the TF, she can quickly find out what test cases may be affected by the modification by passing the file name and the line number of the part to the impact analyzer tool. In doing so she is provided with a list of all potentially affected test cases.

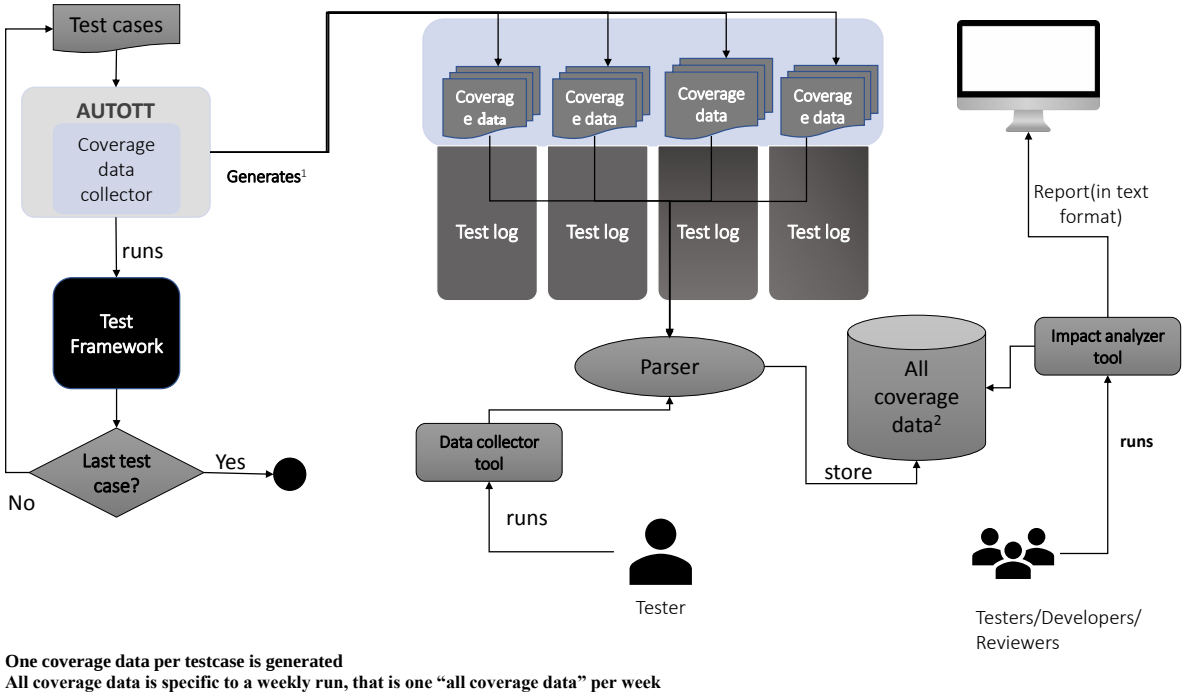

Figure 4.2: The CIA tools and the testers/reviewers interaction with it

[2]Hash table (hash map) is a data structure which implements an associative array abstract data type, a structure that can map keys to values

A training material, which is in Appendix B, was sent to the testers, developers and reviewers in advance before they start using them.

An important note related to this is that the risk of a modification is not quantified by the tool itself. The tool provides a set of potentially affected test cases for a specific input, which is the modified part of the TF, but the developers have to manually verify and quantify the actual impact. A complete set of potentially affected test cases can thus be attained by compiling the results from each individual input.

4.3.2 Motivating examples

In this section we will provide an example of how a maintainer, who wants to modify a specific file in the TF called `platform_packetloss.py`, tries to understand which test cases will be impacted by her changes. First, we briefly describe how the test cases are automatically executed each weekend, then how coverage data is collected and finally how this coverage data is used for impact analysis. At the end of the section, additional uses of the tool outside the intended use are discussed.

On April 6, Like any other weekend runs, the test portal started running the test cases one by one. However, at this time the coverage tool has been integrated in AUTOTT so that at the end of each test case run a coverage file is generated and stored in the test log directory. These coverage files contain both a text output and an html output as shown in figure 4.3.

The HTML report is an easily understandable color coded report where one can check for each test execution which part of the file the test case has used (green color) and which part of the files it hasn't used (red color). The coverage file, which contains line numbers of the TF covered by a test case, on the other hand is not in a human readable format so it has to be changed to an easily understandable format.

One coverage file and one HTML report are generated per a test case and they are stored in the log files of each test case. In order to simplify the utilization of these files, they have to be collected and stored in one file, which is done by running the collector script. There are hundreds of test cases running hundreds of functions, so the size of the coverage file is huge (up to 15Gb). Thus, to easily manipulate and to avoid spending much time in manipulating this file, the final coverage data, which is a combination of all coverage files generated in one weekend, is stored in hash tables.

Finally, the maintainer who wants to modify the TF can check what test cases are affected by his changes by running the ***cia (change impact analysis)*** script as shown in the figure 4.4

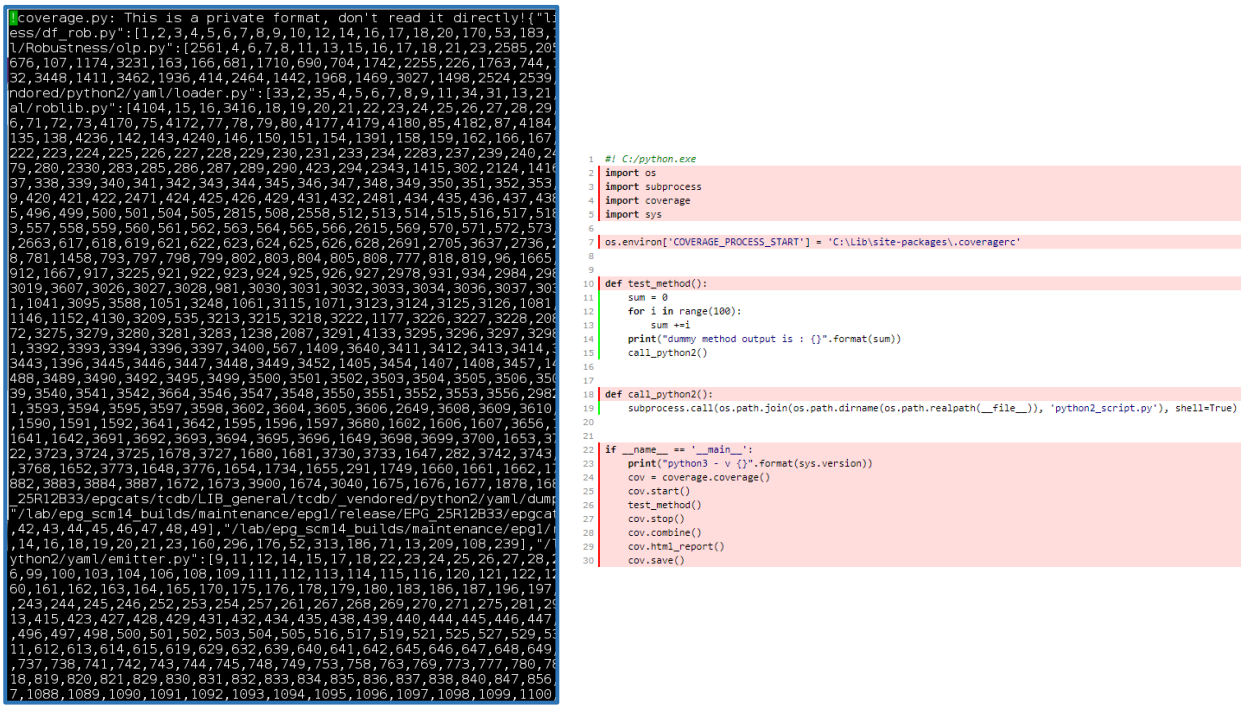

Figure 4.3: The coverage file(code statements of the TF covered by a test case) and the html report generated during the test run

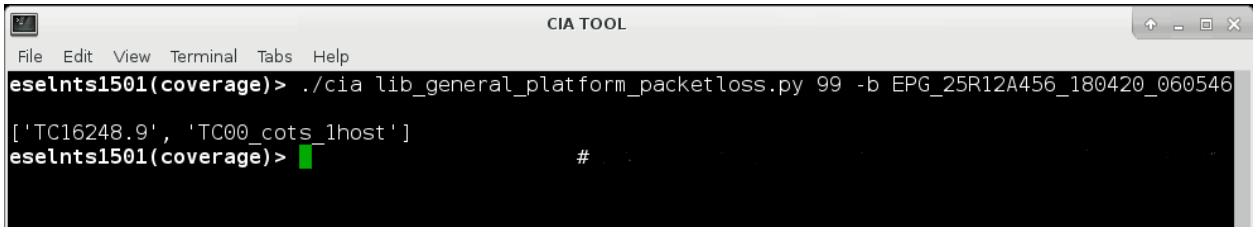

Figure 4.4: Affected Test cases if `platform_packetloss.py` file is changed at line 99

To run the ***cia*** script, two mandatory arguments must be passed as an input. The first one is the file in the TF which is going to be changed and the second argument is the line number where the change is going to take place.
As shown in figure 4.4, the maintainer was informed that test cases `TC16248.9` and `TC00_cots_1host` are going to be affected if `platform_packetloss.py` file is modified at line 99.
Knowing this, the maintainer can make an better informed decision of how to proceed. she may for example decide to run the test cases after finishing his modifications or even modify them as needed to avoid introducing defects.

In addition to the above main intended purpose we have found another way that the tool can be used, namely to identify dead code. The tool provides information about what parts of the TF are used by the regression suite and so one can also use it to find out what parts are not used. If some part of the TF does not have any

coverage information at all, it indicates that it may be dead and by analyzing the change logs one can determine whether or not the piece of code is actually dead.

As mentioned in section 4.2.3, we found in the interviews that dead code is a recurring problem in the TF and so there is a need to implement tools to assist developers in dealing with dead code.

In fact, during the thesis work the industry supervisor tried using the tool for removing dead code. Using the tool she was able to identify a set of seemingly unused files which she then investigated further. After consulting the change logs and the responsible developers, they decided to remove a total of ten files.
This could be a possible future extension of the tool we implement for CIA, where the same tool could provide a complete set of all files that lack coverage information indicating that they should be investigated. However, this is outside the scope of this thesis and therefore we will not be investigating this matter further.

4.4 Evaluation results of the tool

In this section we present the results of the interviews conducted with the users that tested the tool. First we present results regarding the research questions and finally the improvement suggestions.

To ensure the anonymity of the interviewees we refer to them as TOR 1, TOR 2, LSV 1, LSV 2 and XFT and we will refer to all of the using the female pronoun.

RQ1) Is the tool useful for assisting developers in understanding the TF, the testcases and the dependencies between them?

TOR 1 one stated that the tool was not useful to her for understanding parts of the TF because she already knew all she needed to. In her test object there were significantly fewer test cases to maintain and thus less need for such a tool. However, she stated that in a test object with more test cases it may be useful. In fact TOR 2, whose test object have more test cases, stated that the tool would be very useful in this regard especially for understanding how test cases execute. Furthermore she added that such a tool would also be useful when working with legacy code which may be difficult to understand by just reading the code. Lastly, both TORs agreed that this tool would be useful for a newcomer who needs to understand the framework.

Both LSVs stated that the tool would be useful for understanding the code. LSV 1 stated "Having the information about what is truly executed by the test cases is very useful in understanding the code". LSV 2 specifically stated that it helps in understanding the structure of the TF, adding that the tool is extra helpful when working with the TF because of the lack of documentation of those files. Lastly LSV 2 stated that the tool would be helpful for newcomers to the TF but that it

can't replace training.

The XFT stated that the tool would be useful for newcomers but couldn't say how useful it would be as she didn't think about using the tool for this purpose when trying it out. She added that the tool in its current state is not very useful because it doesn't cover a specific type of files called macros [3]. These are not python files and are therefore not covered by the tool. If the tool could cover those it would be useful.

Thus we have found that the tool is helpful for improving understandability of the TF **(RQ1)** but in its current state primarily for newcomers or LSVs. However all interviewees agree that the tool would be helpful if a future extension of it would include coverage of macros as well.

RQ2) Is the tool useful for assisting developers in understanding the impact of modifications they make or are going to make?

For helping in understanding the impact of modifications both TOR's agreed that the tool would be useful. However each voiced similar concerns about the tool. TOR 1 stated that sometimes the impact set too large to manage and in those cases the tool is not very useful at all. In one case she had selected a file that was used by all test cases within her test object and was summarily provided with a list of all of her test cases. TOR 2 stated that while the tool is useful there is a risk that if one were to send such a list of impacted test cases to a tester she wouldn't know what to do. Testers would need to know which files to look into or at least get some more help in that. Thus there is a need to improve the accuracy of the impact set to ensure that it is manageable for all users.

LSV 2 had similar concerns and stated "It depends on how accessible the information is. If it's simple and straightforward, it would be very helpful. The most critical part however are the macros and if those could be covered it would be very useful. If we specify one line of code and we can see which test cases use it, that's the information we are after!".

LSV 1 stated "While it is not 100 percent what we are looking for in terms of information, it is helpful. We want to know what the problems are and this information (the information provided by the tool) may help us navigate to the problems."

The XFT stated that, it is useful for impact analysis but without a scale, meaning that she couldn't quantify how useful it is. "If we are only going to cover python files and files within TF, its not useful enough to spend time on. If it could cover the macros it would be useful but in its current state I see limited usability".

Thus we have found that the tool is useful for improving modifiability of the TF **(RQ2)** but there are some necessary modifications that should be made before the tool is completely adopted.

[3]Macros are text files which are part of LIB general in the TF.

Additional uses of the tool
In addition to the intended uses of the tool i.e. understanding the TF and helping estimating impact, several of the interviewees found other ways to use the tool that were useful to them.

TOR 1 found that the tool allowed her to do more rigorous controls of change requests. By looking at the coverage data from a specific test case she could check whether or not the modified code in a change request is actually covered by the test. She stated that being able to check which code statements were executed by which tests is something she have wanted but not had the tools to do. Furthermore she and LSV 1 found that the tool could be used for identifying and removing dead code by looking at parts with low coverage and then examining change logs and related files that indicate whether or not the code is used.

LSV 2 found that she could use the tool to help verify the correctness of tests. If a newly created test case fails, she could use the tool to find all other test cases that used the same parts of the TF. If they all passed she could often assume that the error was in the test case rather than the TF.

4.5 Discussion

In this section we discuss the above presented results and the lessons learned from the study. From the interviews we identify a strong recurring theme regarding both RQ1 and RQ2. The tool is useful for improving both understandability and modifiability but it needs modifications to make it truly useful for all involved stakeholders.

The two most important modifications are i) adding coverage of macros and ii) improve the precision of the impact set. If these modifications can be made, all interviewees state that the tool would be useful in their current process both for improving understandability and modifiability of the framework.

However, these are not trivial modifications. In fact coverage of macros was intentionally left out at the start of the project, on behest of our supervisor, because of the difficulty of covering them. The macros are not python files but text files which are executed in a different way than the test cases. Additionally they have much more complex dependencies as stated by the XFT "A test case may be using macro that uses another macro which is using a sub-macro that another test case is using as its main macro. Those things you don't see when you do the execution, you only see what python code was executed. You see the resulting TCDB file which is a copy of all the variables and macros you have created but that's just the end result, you don't see how you got there." Thus covering them requires an entirely new approach which would have to be added to the tool. Furthermore, as far as we know these macros and the way they are used is unique to Ericsson's framework and as a result any tool made specifically for this framework that covers macros will likely not be

applicable elsewhere.

Improving the accuracy of the impact set is also quite difficult. There are two ways that it could be done but each is problematic. First one could improve the tool to actually quantify the impact of modifications rather than just showing potentially affected files. By quantifying the actual impact the impact set will become much smaller as we currently show all test cases using a file or code statement and not test cases whose function is impacted by the modification of those files or code statements. Naturally this is much more complex to do as it requires an additional process after the current impact set is estimated to calculate which test cases are actually affected by the modification.

Another way to reduce the size of the impact set is to allow exclusion of certain parts from the impact set. The exclusion criteria could be to only show parts with a certain coverage level or manually excluding entire sets of test cases. While such approaches may be very effective in reducing the size of the impact set there is a great risk that actually impacted test cases are excluded.

Lessons learned about coverage of test systems
The results indicate that there is merit to the idea of providing coverage information of test cases, with respect to the test system, to assist maintainers in estimating impact and understanding a framework. A key lesson is that for such tools to be useful, they must provide information on all parts of the test system or at least the parts of interest. Thus it is of interest to identify which parts of a framework are of interest to cover and which challenges they present before implementing tools.

Not covering all parts of the test system introduces several risks in the use of such tools. The XFT stated "Low coverage of parts in a test system is not necessarily bad. Some tests are only executed very rarely like once a month or even more seldom. In addition there are manual test cases that are not run by the automation tool and because this tool gathers its data from the automation tool manual test cases are not covered. This means that there are parts of the framework that will always have low or no coverage at all and if someone was given that information they may think that the parts are unused and try to remove them."

Furthermore, both the XFT and the LSV expressed concern that showing coverage in the context of a test system may be misleading because of the difference in what coverage means in a test system compared to an SUT. In an SUT coverage shows how much of the code is tested and thus low coverage indicates insufficient testing. This is not the case in a test system as its tests are for an SUT not for the test system itself. The coverage in a test system only indicates which parts of the system are used by the tests.

Because of this difference in meaning, coverage may behave differently in a test system as well. A component of an SUT may be tested for different purposes, for example functional testing or stress testing and in either case the same or similar

parts of the SUT will be executed. However, this change in testing may cause a complete shift in the coverage of the test system as different parts of it are used for the different tests. Thus coverage in a test system will vary to a greater degree depending on which test cases are executed compared to its SUT. The XFT stated "In Ericsson's case the test priorities are usually different in each week and so different tests are selected for execution. Thus the coverage of their framework will also vary greatly from week to week which may cause confusion and frustration for maintainers if they don't understand why this is.

Thus, when adopting tools that use coverage in the way ours does there must be sufficient training and instructional material for maintainers to understand and properly use the tools.

Different maintainers have different needs
Despite our small number of interviews, we were able to see some distinct needs for LSVs and TORs with regards to what the tool provides. The TOR needs the tool for understanding and controlling change requests that are sent to them. The LSVs on the other hand need information that can help them when working with the test cases for example to localize faults, help them understand the test cases or to verify the correctness of test cases.

Because of the difference in needs it may be desirable to elicit further information about the requirements from each group with regards to tools for CIA before making modifications to the tool. It may very well be the case that the needs of one group contradict the needs of another or that a necessary modification/feature for one group may impact the usefulness of the tool for another. Furthermore, as mentioned three interviewees found additional uses of the tool, which suggests that there are other needs that may be met with tools providing similar information. To fully capture the needs of each role one should start with thorough requirements elicitation for each role. By creating an overview of these needs one can determine whether or not to implement tools for each role or just one tool that may have specific features for specific roles.

4.5.1 What are the challenges in implementing a tool for CIA in a test framework?

To answer our third research question we conduct a retrospective analysis of the challenges we have faced during the implementation of the tool. The purpose is to identify possible challenges that one should consider when implementing such tools for a testing framework.

Choosing frequency of data elicitation
One of the first issues we ran into was when and how often to elicit coverage data since it has a significant impact on the tool in terms of accuracy. If ones organization has a very high frequency of changes, one has to ensure that the data is collected

often to avoid using obsolete data.

As described in section 4.2 the context of Ericsson's framework limited us to relying on coverage data which we elicited once a week because of the high costs of executing the regression suite. Even though Ericsson runs regression test cases on weekends, our tool has a capability to elicit coverage data at anytime as long as they are regression test cases run by the test portal.

Thus, practitioners should be aware of the recency of the data they are using as it could affect the accuracy of the tool.

Choosing granularity of object and impact set
The second major challenge we encountered was deciding what levels of granularity the object and impact set should have. The granularity of these are directly related to what the tool has to achieve. Furthermore, achieving a more precise granularity often implies additional efforts and the challenge presented is to balance the needs of precise granularity versus the required efforts.
In our case we had to achieve a granularity on the method level for the object at least because maintainers needed to see for specific modified parts of the TF what could possibly be impacted. However because the efforts of achieving a granularity on the code statement level were very small we decided on a finer granularity than absolutely necessary. As a result maintainers can even more accurately determine the impact as they know precisely which code statement they are analyzing as opposed to which method.
For the impact set we needed to see which files were affected and so the impact set needed a granularity of file level at least. Improving the granularity further turned out to be very complex because of our selected approach and how tests are executed by the automation tool. Because of this and the fact that our priority was to determine which test case files are affected by modifications we decided to not spend effort trying to refine the granularity.

Covering sub-processes
The third challenge we faced was inability to cover files in the TF run by sub-processes. The coverage module used can cover files which are run by a sub-process, but achieving the module to do that involves extra steps which are not trivial, which are:

- First, upgrade-to/install the 4.3 coverage version or later.
- Second, configure the module as specified in the module help page to support sub-processes.
- Lastly, change the system python environment setup. To cover all files run in sub-processes, the coverage module should be started before a test execution starts. This is done by modifying/creating the *sitecustomize.py* file to start the coverage module before anything else. *sitecustomize.py* is one of the files a python interpreter calls before anything else.

Thus, practitioners should be aware of the architectural complexity of their test framework in terms of how sub-processes are used. Not considering this may result

in inability to elicit all coverage data which in turn affects the reliability of the tool.

Covering all necessary parts of a framework
The fourth challenge we encountered was covering all necessary parts of the framework. We decided on leaving coverage of macros as a later extension to the tool we implemented. However, the implication of doing so was that the usefulness of the tool was significantly reduced.

This shows that one must be aware of which parts of the framework are critical to cover as well as the challenges related to doing so. If critical parts are missed the tools usefulness may be reduced and unless there is a plan for how to achieve coverage of such parts it is a very real possibility that the tool won't achieve its goals.

Considering the expertise of the users
The last challenge we were faced with was considering the expertise of the users of the tool. Any tool for CIA that is implemented must be understood by its users. If the information it provides is inaccessible due to the complexity of what the tool shows the tool will likely not be used.

In our case we found that the impact set is sometimes to large to handle for maintainers. They needed more precise information or a sensible visualization to be able to use the tool effectively. Furthermore, as mentioned regarding the lessons learned about coverage of test systems we also had trouble with the term coverage causing confusion as the term had a different meaning in the context of our tool. Our suggested solution to this issue was to provide maintainers with basic training to understand the tool and how the meaning of coverage changes in the context of the tool.

This shows that one must be aware of the expertise of the people who will use the tool. The impact of not taking this into account may be that the tool is completely unusable by its target users.

5

Results: Facilitating finding reviewers

In this chapter we present the findings of the literature review we conducted to answer our third research question addressing the issue of finding reviewers. First we investigate the option of adopting an RR tool by identifying key questions one should answer before choosing a tool to implement. Then we provide a suggestion for how to change Ericsson's code reviewing process by applying practices and models from literature that facilitate reviewer identification. Lastly, we answer the questions of the guidelines in the context of Ericsson, present a list of available RR tools and provide a recommendation about which tool to adopt based on the answers.

5.1 Guidelines for choosing a tool for finding reviewers

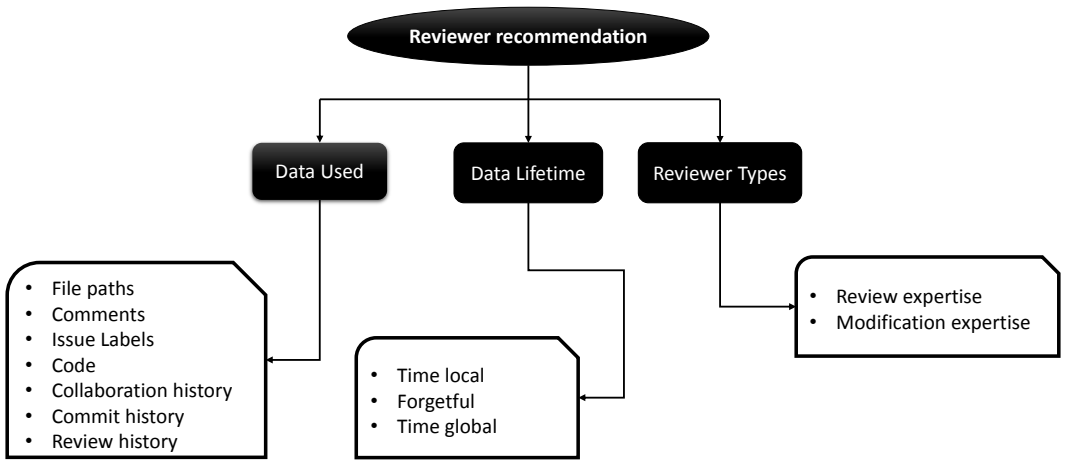

Figure 5.1: Framework to follow when adopting a tool for RR

By analyzing and comparing different RR tools we found that there are three main areas where techniques differ as shown in figure 5.1: i) The data they use, ii) the lifetime of the data iii) the type of reviewer they recommend. In the following sections we describe these areas and how the different choices related to them may impact the choice of technique.

5.1.1 What data do we have that can be used for RR?

In answering this question one can quickly determine if there are any existing tools that can be used. As mentioned in section 2.3.1, most RR tools rely on historical data acquired from issue tracking systems, or similar systems. Therefore one can start by analyzing one's own systems to identify what data can be acquired from them.

Depending on the kind of data one wishes to analyze, there may be additional requirements one must comply with for an RR tool to provide accurate results. For example, when using an algorithm based on analyzing file path similarity there should be a clear standard for all files i.e related files should be in similar paths. Such algorithms determine the expertise under the assumption that files with similar paths are closely related by their function [65].

5.1.2 What is our data lifetime?

How long the data, that the tool uses, is stored may affect the choice of RR tools as old data may become obsolete and impact the accuracy of certain tools. Lifetime of data is a significant factor in RR approaches as expertise degrades over time [63, 65]. Some approaches take this into account and some don't and we can distinguish between three types of algorithms used in RR approaches which we describe below [65].

1. Time local algorithms - only consider data that was generated shortly before execution.
2. Forgetful algorithms - consider all data available but weights data based on recency.
3. Time global algorithms - consider all data available equally.

Time local algorithms are not affected by old obsolete data as they exclusively rely on very recent data, for example, the line 10 rule which recommends the developer that last modified the code [66]. The biggest issue with such algorithms is that they tend to be much less accurate than forgetful and time global algorithms [65] likely because of the very limited set of data they use.

Both forgetful and time global may increase in accuracy over time as their input data grows but forgetful algorithms are more accurate as they consider equal amounts of data but takes its recency into account. Time global algorithms may lose accuracy if there are changes in the project because old data, that may have become obso-

lete, is still taken into account. Hannebauer et al. explains this with the following example "Two reviewers that were active for two years will both be recommended with equal probability, even if one of them was active only five years ago and then left the project and the other is still active. This effect decreases their prediction performance over time and may also affect forgetful algorithms that do not properly forget obsolete data" [65].

Thus before choosing an RR tool one should consider the lifetime of one's data. If data is never removed and the project undergoes significant changes, it may be necessary to use a forgetful algorithm to get accurate results while if data is only stored for a shorter time, one might be able to achieve equal results with a time global or time local algorithm.

5.1.3 What kind of expertise are we interested in?

There are two kinds of expertise that can be analyzed, review expertise which is determined by analyzing a person's previous reviews and modification expertise which is determined by analyzing a person's previous work [65]. There are benefits and drawbacks with each and the the choice of which to use depends on how well these benefits and drawback suit one's needs [65].

Techniques that rely on review expertise have been proven to be more accurate in their recommendations. They only consider those who have already performed reviews excluding all who do not have review rights. However, this also means they exclude potentially competent reviewers if these have not done any reviews, which prevents new reviewers from being considered by the tool until they have made at least one review [65]. This could easily lead to the tool always recommending a small set of reviewers who quickly become overburdened.

Modification based algorithms have a similar problem because such algorithms only consider people who have made modifications to the code and will therefore exclude potentially competent reviewers if they have not modified the code. Additionally modification expertise algorithms may recommend developers without review rights [65]. However it is possible to take this into account and Hannebauer et al. states that if such algorithms filter out reviewers without review rights before making recommendations they will likely achieve a greater accuracy [65].

An advantage with review expertise algorithms is that sometimes it might not be desirable for the one with the best modification expertise to do reviews, as they may be better developers than reviewers and might be best used as developers. Because such algorithms do not depend on the competence one has working with that code but rather the competence in reviewing it, this is less of an issue than for modification expertise algorithms[65].

A significant drawback of relying on review expertise is that tools that rely on it create cycles as their recommendations affect the data they use in their analysis as

shown in figure 5.2. When the tool makes a recommendation and the recommended reviewer performs a review the new review will be added to the review history and the next time the tool makes a recommendation it will be even more likely to recommend the same reviewer.

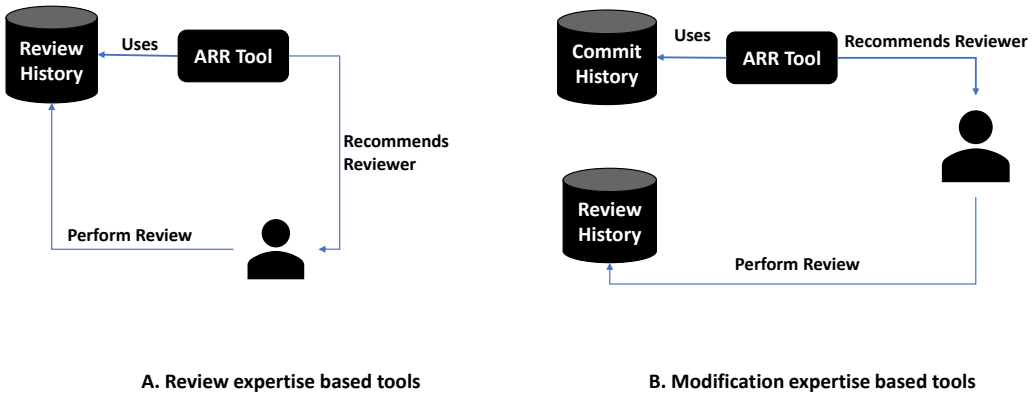

Figure 5.2: (A) The cycle created in review expertise algorithms, (B) The non cycle in modification expertise algorithms.

Modification expertise based algorithms don't have this problem as the output is not used by the algorithm itself in its recommendations. As shown in figure 5.2 when the tool makes a recommendation and the reviewer makes a review, it doesn't create a positive feedback loop as the algorithm uses the commit history for its recommendations rather than the review history. This is the one of the main benefits of modification expertise.

While it is possible to use both types of expertise to determine the expertise of a reviewer, doing so may have adverse results on an algorithm's accuracy. Zanjani et al. found in his study that an algorithm using commits and reviews for its analysis provided less accurate results than one just using reviews for its analysis [63].

However, there are reports showing that combinations of approaches in RR tools, for example using review expertise and collaboration history or common interests and modification expertise, may increase the accuracy of recommendations. Yu et al. found in his research that an algorithm that combined analysis of expertise and common interests among developers i.e social factors was more accurate than those who only used one approach [10]. Additionally many newer algorithms that outperform older ones rely on more factors than just expertise including frequency and recency of collaboration [62] and implicit relations in review data [67]. Those tools

that do rely solely on expertise often use several metrics like frequency and recency of reviews/commits to enhance their precision like the tools presented by Zanjani et al. and Xia et al.[63, 67].

This suggests that there may be benefits using the data of both modification and review expertise in the same tool but one should carefully consider how the data is used and for what purpose. While there is a risk of degrading performance when both are used for determining expertise, they could be used to in the same tool but for different purposes. For example, one could analyze previous work to determine the expertise and previous review comments to determine common interests similar to what Yu et al. did in their study[10].

5.2 Suggestion for Ericsson

In this section we provide a suggestion for how Ericsson can apply these findings to change their current process to facilitate finding reviewers. First we present what practices may be applied and how responsibility should be addressed. Lastly we answer the three questions we have provided in sections 5.1.1, 5.1.2 and 5.1.3 in the context of Ericssons TF.

5.2.1 Proposed changes regarding practices

To address the issues of improving response times of reviews, the usefulness of feedback and finding reviewers MacLeod et al. presents several practices for code authors and code reviewers, as shown in 2.1.

Most of the practices of table 2.1 are already being applied at Ericsson to some extent including A.i-iii, R.i, R.ii, R.v and R.vi. However, their use is not mandatory as there are no actual guidelines for how to do code reviews. Thus a way to ensure that the practices are more strictly adhered to and to adopt the new practices is to implement such guidelines. These guidelines should include all practices for authors and reviewer presented in 2.1.

The new practices that should be included in such guidelines are: R.iii (using review checklists), which may be extended to authors as well and A.iv (testing and analysis of code). The inclusion of A.iv implies that tools for analysis of code must be implemented and a good start would be using and extending the tool for CIA implemented in this thesis. Lastly, to further address the lack of documentation one could include instructions for how to document code in the guidelines as well. To facilitate adoption of new practices and establishing a sustainable reviewing process, Ericsson should also adopt the organizational practices presented in 2.2.

However, while the practices we found address some root causes of the issue of finding reviewers, like finding relevant documentation and transferring knowledge, they do not address the lack of responsibility of files or the lack of tools for finding reviewers. Therefore we suggest addressing this issue by first by creating a clearer structure of

responsibility, to ensure that there is always at least one responsible developer for each piece of code and secondly to implement a tool for reviewer recommendation. As both solutions are complex we provide a more extensive explanation for how to implement either in sections 5.2.2 and 5.2.3 respectively.

5.2.2 Proposed change for responsibility

Because of the close relationship between code ownership and code responsibility we use Nordberg's work, which we presented in section 2.3.2, to find out which model suits Ericsson best at this time. Note that code ownership should be defined as having responsibility for understanding, maintaining and reviewing a piece of code.

Because the testing framework is not a newly starting project but a fully functioning software with many involved developers it is inadvisable to start from the beginning with the product specialist model with a single code owner. The appropriate model to adopt at this point would be subsystem ownership, with individual owners for every sub-system, as there are many functioning components and sub-systems in the framework. Thus, Ericsson should start by assigning responsibility of such components and sub-systems to teams or individuals that are working with them. Finally to keep track of who has responsibility of what code one could for example create maintainer files for different components that explain who is responsible for the component.

5.2.3 Application of guidelines

In this section we answer the questions presented in the guidelines of section 5.1 in the context of Ericsson's TF. Afterwards we provide a necessary information to Ericsson for making decisions about how to proceed with RR tools.

What data is available for analysis?
When it comes to data, Ericsson's code review tool can provide the required data for almost any RR algorithm available. However, because of their current file structure, in which unrelated files may have similar paths, it will be difficult to apply RR tools based on file path similarity because, as mentioned in section 5.1.1. Therefore the file structure must be addressed before any tools based on file path similarity can be implemented.

The second issue is that because of the lack of guidelines for how to perform reviews, maintainers are free to perform many actions that may affect the validity of data used by RR tools. For example, developers could tag a large amount of reviewers for a commit, which may give the impression that all reviewers contributed to the issue. Thus if an RR tool would take that data into consideration it's accuracy of recommendation would likely be negatively impacted. Thus we find additional reasons why such guidelines should be implemented.

Both these issues should be addressed before RR tools are implemented.

What is the data lifetime?
Currently, there is no limit to how long data is stored and thus there is data from when the system was first introduced. Because of the lifetime of the data there are two issues that one must consider.

First, there is a significant risk that developers found in the code reviewing system are no longer with the company as the data is several years old and many reviewers have been employed by or left the company in that time span. This means that any RR tool used at Ericsson must be able to determine if a reviewer is still in the company or not.

Secondly, because of the high frequency of change at Ericsson, specifically within the TF, old data becomes obsolete faster. If such data were to be included by an RR tool it would likely have a negative impact on the accuracy of the tools recommendations. Therefore older information should not be analyzed or at least weighted lower to avoid this issue. Because of this any RR tool would have to use either a forgetful or time local algorithm to not lose accuracy in recommendations over time.

What expertise are we interested in?
As mentioned in section 4.1, a finding from the interviews is that there is a perception that knowledge of code only exists within teams and with certain individuals. This is a hindrance in finding reviewers as many believe there is no one that has the necessary expertise to perform insightful reviews.

Because finding reviewers have been a problem at Ericsson for some time, the existing data of previous reviews may be misleading. It is entirely possible that the actual reviewers have not been the most suitable ones and therefore modification expertise may yield better recommendations early on as the recommendations will be of people who have actually modified the files, or at least similar files. However, only using modification expertise will likely leave out a large number of competent reviewers as there are a significant amount of people at Ericsson who perform reviews but do not produce code.

While algorithms using review expertise risks leaving out reviewers if they have not performed reviews, the impact of such an exclusion is likely lower as most who have not performed reviews are either very new or don't have review rights. Furthermore, review expertise is much more prevalent in recent algorithms that significantly outperform older algorithms. Therefore, we suggest using a tool based on review expertise. We choose not to recommend using an algorithm that combines both types of expertise because of Zanjani et al.'s findings we presented in section 5.1.3.

5.2.4 What should be done regarding RR?

Thus, after applying the guidelines at Ericsson we identify two actions that should be taken. i) Establish a clear standard for files such that similar or related files should have similar paths. Code that is unrelated should be extracted into separate folders. In doing this Ericsson lays the foundation for using any RR tool that relies on file path similarity. ii) Implement a tool for RR that uses a forgetful and/or time local and a review expertise based algorithm.

Regarding the tool itself we suggest using existing algorithms and tools and modifying them to suit one's needs if necessary. In figure 5.1 we present ten different tools, showing what data they use, what time type they have and what kind of expertise they use. As mentioned in section 2.3 information from CIA tools can be very helpful for finding reviewers because you essentially have more accurate input data to the RR-tools. However, none of the tools in table 5.1 use CIA as a means to help for finding reviewers.

Tools	DATA								TIME			Expertise	
	File paths	Comments		Issue Labels	file	Collaboration History	Commit History	Review History	Time local	Forgetful	Time Global	Review Expertise	Modification expertise
		Frequency	Recency										
REVFINDER	X				X			X			X	X	
RevRec	X	X	X			X		X		X		X	
WRC					X			X		X		X	
PR-CF		X	X					X		X		X	
FPS	X				X			X		X		X	
Line 10 rule							X		X				X
Expertise Recommender				X			X		X				X
Number of Changes							X			X			X
Expertise Cloud	X				X		X			X			X
CHRev		X	X		X			X				X	

Table 5.1: Available tools for reviewer recommendation

Based on the answers to the guidelines questions we believe that the three most suitable techniques are WRC, RevRev and PR-CF. All three are recent tools using forgetful algorithms and review expertise that have a much higher accuracy than all other presented tools. In table 5.2 we provide more detailed information about these tools. We excluded all algorithms based on modifications expertise as all such algorithms we found were significantly older, most being from research before 2010, and less accurate.

During the study we investigated several other tools and the complete list of investigated tools can be found in appendix A.3. The tools that are not presented here are tools that were either proven inferior to at least one the tools in the table or used similar techniques to one or more of the presented tools with similar accuracy.

Tool	Input	Time Type	Top 1 precision	Size of evaluation	Systems used in evaluation
WRC [65]	File, recency and frequency of reviews	Forgetful	N/A	60.000 - 120.000 issues and 300-3000 reviewers	Firefox, ASOP, Qt and Openstack
RevRec [62]	File paths, collaboration history, frequency and recency of comments	Forgetful	49-59	5000 - 23000 issues. 94-202 reviewers	Openstack, Qt and android
PR-CF [67]	Reviews, comment network, implicit relations	Time local and Forgetful	60-91	3.000 - 15.000 issues and 200-1500 reviewers	angular, netty, saltstack/sal, ipython and symfony

Table 5.2: Table showing three recent tools for RR

WRC calculates and sums up reviewers knowledge about a specific issue by analyzing review experience for similar files at the time of their reviews. It uses Expertise Explorer, an OSS platform for running expertise analysis, which is publicly available on Github. WRC was evaluated on the largest set of issues and with the largest amount of reviewers but did not use precision or recall to determine its accuracy which makes comparison to the other techniques difficult. However WRC has been proven to outperform FPS which was the second best algorithm in the study presenting PR-CF with a precision of 66 to 81 percent for top 1 recommendations. Thus it is likely that WRC will have a comparable accuracy in terms of precision and recall.

RevRec identifies the file of the modified source code and then looks at who has reviewed that code in the past while taking into account the reviewers previous collaboration with the author. Revrec was evaluated with by far the smallest amount of reviewers in the systems it used for evaluation. Both WRC and PR-CF were evaluated on systems with more than 1000 reviewers while the largest system RevRev was evaluated on had 200 reviewers. Thus, if this tool was to be applied in a system where there are more than 200 reviewers there is no way of knowing how well it will perform. It was however evaluated on some systems that WRC was evaluated on including Openstack and QT which makes comparisons easier.

PR-CF uses latent factor modeling and neighborhood methods to capture implicit relations and it takes recency of reviews into account. It is both time local and forgetful as it has two tiers in its algorithm. It was evaluated on completely different projects and it had a much smaller set of issues to work with but more reviewers than Revrec. PR-CF seems to have a higher precision than both other algorithms but this could very well be caused by the differences in the evaluation. PR-CF also has the largest span of accuracy of 31 percentage points compared to Revrec whose span was 10 percentage points. This suggests that the accuracy of PR-CF may be affected by the project it analyzes to a greater extent than the other two.

5.2.5 Results of evaluation of recommendation

In this section we present the results of the interview we conducted to evaluate the usefulness of the guidelines for choosing RR tools. The objective of the evaluation was to determine if the guidelines provided are useful for understanding RR tools. Once again to ensure anonymity we will refer to the interviewee by their role or use the female pronoun.

The interviewed TOR stated that the guidelines are good and provide a quick overview of such tools. She admitted to not having any previous knowledge about RR tools.

While she found the first question rather trivial the time types and expertise types related to the following two questions were of great interest. She stated "That some tools consider all data is useful to know for us, because here at Ericsson people move around a lot and the poor guy who worked on something two years ago doesn't want to be bothered with reviewing those things. So I can see that we would want a forgetful algorithm". Furthermore she stated that understanding what kind of expertise these tools calculate helps to figure out if such tools are of interest. She goes on to state "When I have a problem I need to find someone for, I might be looking for someone who is knowledgeable in python for example, or someone who knows the domain very well, so just modification or review expertise may not be what we are looking for.".

By having the guidelines presented she understood some of the benefits and limitation of RR tools. She stated that the guidelines gave her enough information to make a case for whether or not they should investigate RR tools further and when asked about what Ericsson's next step regarding RR tools would be she answered "I think it would be of interest to investigate further knowing this, and tools we find should be implemented and tried side by side to see which is the most useful. Only then can we truly see if the tools are useful enough to be adopted".

To quantify the perceived benefit of RR tools she added "If we investigate further, it should be done by a summer worker or thesis worker because it's not useful enough to warrant spending significant paid efforts on it".

5.2.6 Discussion

The results indicate that the guidelines are useful for understanding RR tools as the interviewee was able to quantify the usefulness of such tools and make a decision to investigate further despite having no previous knowledge about RR tools. However, from the interview we found that there are two additional informational needs when choosing RR tools that should be included in the guidelines.

The first is to provide deeper explanations for how tools calculate their expertise as well as more distinct classifications than review or modifications expertise as this greatly affects the usefulness of such tools in different contexts.

The second is to distinguish between approaches that are just algorithms and those that are supported by tools. When the interviewee was told that out of the three recommended tools only one of them had tool support she immediately dismissed the other two as they would require much larger implementation efforts further arguing that the usefulness of such tools wouldn't justify the cost of developing them. Thus there is a need for research that summarizes the available approaches similar to Li et al.'s work in CIA [1] and separates tools from algorithms.

The fact that modification or review expertise may not be of interest is an observation that should be investigated further. If practitioners are interested in other kinds of expertise than modification or review expertise, research on RR tools should focus on creating tools for those as well.

We found no RR tools that were evaluated with actual users and it is very possible that the research in RR does not match the needs or practitioners. These needs must be investigated further so that any RR tools that are developed actually match the needs of practitioners.

Thus, the answer to **RQ5** that we provide may not be complete. In addition to the guidelines the following should be considered when choosing RR tools: i) Whether or not an approach is available as a tool or if its just an algorithms and ii) if the kind of expertise the tool calculate is of interest. However, because of the lightweight evaluation it is a very real possibility that there are other factors that should be considered as well. Thus further research on this topic is required to fully answer **RQ5** and the guidelines provided are a starting point for this research.

6
Threats to Validity

In this section we address the potential threats to validity in our thesis work.

6.1 Construct validity

Construct validity, according to Runeson, means "to what extent the operational measures that are studied really represent what the researcher has in mind and what is investigated according to the research questions" [40].

In our evaluation of the tool we interviewed a small set of users, and the small size of our sample is a threat to our validity. We addressed this threat by having more in depth interviews with each user to fully capture the experiences of each user. Our process to ensure the validity of any instruments created for the thesis is describe in section 3.5.3. Another threat regarding the interviews is that the interviewees may not have had enough time to try out the tool. To address this we tried to prepare the interviewees as best as possible by providing them with training material beforehand and booking the interviews well in advance. Lastly we made sure to answer any questions that the interviewees had during the interviews regarding the tool, its purpose and its process so that the interviewees could get a full understanding of the tools capabilities and limitations.

Since we relied on manual searches in our literature reviews and did not use any automated tools for searching larger databases a threat to the validity of the review is that we may have missed key research reports.

To address this threat we relied heavily on the research that we found by extensively studying the related work reported by experienced researchers and using their findings in a snowballing approach. Because much of the research we found was very recent, the most recent being from late 2017, it is likely that relevant recent research would have been included in such sections. Furthermore we have had a highly iterative approach where we have been using existing resources to assist in finding new keywords to perform more relevant searches.

The evaluation of the recommendation and guidelines we provide was very lightweight as we only interviewed one representative from Ericsson. We addressed this issue by choosing the representative with great care and the interviewee was one of the

most experienced and knowledgeable employees within the relevant area of Ericsson. Thus, the interviewee was likely the best suited to assess the usefulness of any suggested guidelines for choosing tools that would be adopted into the working process.

6.2 Internal validity

"Internal validity is of concern when causal relations are examined. When the researcher is investigating whether one factor affects an investigated factor there is a risk that the investigated factor is also affected by a third factor. If the researcher is not aware of the third factor and/or does not know to what extent it affects the investigated factor, there is a threat to the internal validity" [40].

In our first meeting with our industry supervisor, we were presented with several issues they faced in the current testing framework that they wished us to address. This might have introduced a bias towards what solutions we thought were available and what parts of literature we would investigate. We mitigated this threat by in the first set of interviews try to identify what challenges the maintainers were actually facing, and only once we had compiled the results did we decide on what to investigate.

Because of the small number of interviews there was a risk that we would miss important factors that cause problems in the working process. The same risk is also present during analysis of the results as we may have misinterpreted results or missed important factors. These are threats that concern all sets of interviews and the analysis of each.

To address these threats the interviewees were selected with care with help from the industry supervisor such that all interviewees were experienced in working with the framework and that each key group that interacts with the framework was represented. To avoid misinterpretations during the interviews we spent extra time ensuring that each interviewee understood what they were being asked. As a result the length of the interviews varied significantly as some interviewees needed much more time to fully understand the questions and frame a response. All interviews were transcribed by both researchers individually and later compared to check for inconsistencies and bias. Any uncertainties that arose were resolved by emailing or directly talking to the corresponding interviewee to clarify result. We found strong recurring themes in the interviews which indicates that the findings were at least relevant.

6.3 External validity

External validity concerns the extent to which it is possible to generalize the findings and to what extent the findings are of interest outside the investigated case. For

case studies, the intention is to enable analytical generalization where the results are extended to cases which have common characteristics and hence for which the findings are relevant, i.e. defining a theory [40].

When creating the guidelines for choosing a tool for finding reviewers we made generalizations based on literature. Thus, the applicability of the guidelines is dependent on these generalizations. To ensure the validity of our generalizations we relied heavily on existing research, summarizing findings from several researchers. Our process to ensure the validity of findings in literature is describe above section 6.1 about construct validity. Furthermore, we found strong recurring themes among the tools we studied which we based the generalizations on.

Some of the findings in this thesis are very specific to Ericsson since it was the source and context where the study was applied. For example the macros and the need to cover them, the different needs of each role and the challenges we encountered when implementing the tool. More generality of the results can be achieved in future work as different aspects of our tool and guidelines are applied in different contexts.

6.4 Reliability

Reliability concerns the extent to which the data and the analysis are dependent on the specific researchers. Threats to this aspect of validity is, for example, if it is not clear how to code collected data or if questionnaires or interview questions are unclear [39].

Before conducting the initial interviews we studied documentation of the current working process and the framework itself to ensure that we had a good grasp of the process in theory. Additionally our industry supervisors held several sessions during which they described and explained the process in theory.

To ensure that any of the data we elicited from either interviews or literature was not influenced by our own bias we always discussed, and reviewed each others findings in literature and both researches were present during all interviews.

7
Conclusion

In this study we have investigated how maintainability of a testing framework could be improved by adopting a tool for CIA and by investigating both practices for improving code review processes and tools for reviewer recommendation.

Thus we divide the thesis in two parts i) improving understandability and modifiability of the TF, which are two characteristics of maintainability, by implementing a tool for CIA to use in maintenance tasks and ii) investigating ways to facilitate finding reviewers by searching in literature. Thus the first part of the study answers the following research questions:

> RQ1: To what degree can modifiability of a testing framework be improved by using a tool for CIA?
> RQ2: To what degree can understandability of a testing framework be improved by using a tool for CIA?

We tried to answer these questions by implementing a tool for CIA based on eliciting coverage information of the test system as tests for the SUT are executed and then having experienced maintainers from different roles try using the tool. Finally we evaluate the usefulness of the tool by conducting semi-structured interviews with the maintainers.

From the evaluation, we can conclude that modifiability and understandability of the testing framework can be improved by using a tool for CIA. Thus, we have found that there is merit to the approach of using coverage information on a test system for CIA. However, we also found two necessary improvements that must be made to the tool. First, all parts of the test framework must be covered and not just the TF. For example, for the tool to be fully adopted at Ericsson, the macros should also be covered. Second, the impact set of the tool is sometimes to large to manage and must therefore be reduced to a manageable size by improving the tools accuracy. From this we learn that that there are key things to consider when implementing such tools including considering the varying needs of different roles and the difficulties and benefits involved in identifying and covering all necessary parts of a framework.

Additionally, a short training should be given to the users of tools like ours, because using coverage for change impact analysis may be confusing for maintainers who do not understand the the difference of coverage of a test system compared to coverage

of its SUT. The users of the tool were familiar with coverage in the context of an SUT and in this context coverage refers to which parts are tested or not. Thus maintainers may for example think that high coverage is necessary. However coverage in a test system only shows which parts are used and low coverage is not necessarily a bad thing as it only means that some parts are less frequently used. Thus, training should be given to users to avoid such kind of misconceptions.

To answer RQ3 we conducted a retrospective analysis of our process developing the tool and identified all significant challenges we faced as well as the impact they have or may have had on the tool. We identified five major challenges including: i) choosing frequency of data elicitation ii) choosing granularity of object and impact set iii) covering sub-processes iv) covering all necessary parts of the framework and v) considering the expertise of the users. Each challenge, our way of overcoming them and their potential implications are described in section 4.5.1.

The second part of the study revolved around answering the following research questions:

> RQ4: What are the current best practices in literature regarding code review and code ownership that facilitate finding reviewers?
> RQ5: What should be considered before adopting tools that support stakeholders in finding appropriate reviewers for their code commits?

To answer RQ4 we studied literature to find the best practices to facilitate finding reviewers. The main findings are from research by MacLeod et al. [12], supported by findings from research by Bacchelli et al. [13], Tao et al. [14], Rigby et al. [25], Bosu et al. [60] and Fu et al. [61].

First we identified practises that may indirectly help in facilitating reviewer identification by improving response time and providing authors and reviewers with additional necessary information. Secondly we investigated practises that may directly help in facilitating reviewer identification. All identified practises are presented in section 2.3. We describe how these practises should be applied at Ericsson in section 5.2.1. In short most practises should be included in a set of guidelines for all involved stakeholders at Ericsson describing how code reviews should be conducted. How these practises are best applied will likely vary depending on the context and anyone trying to modify their code reviewing process in accordance with the best practises should be aware that there may be unique aspects in their respective environment that affect how practises should be adopted.

The Last research question, RQ5, is answered by analyzing literature on RR tools in order to provide guidelines of what to consider before adopting RR tools. We found that there are three questions one must answer in order to specify ones requirements on such tools and identify potential tools that meet these. The questions are i) What data do we have available for analysis? ii) What is the lifetime of our data? and iii) What kind of expertise are we interested in?

The guidelines were evaluated through an interview with one of the most experienced individuals working with Ericsson's testing framework where we presented the guidelines and applied them at Ericsson. From the evaluation, we can conclude that the guidelines are useful and covers most of the important criteria that should be considered before adopting a tool for finding appropriate reviewers. However, we found two things that should be added to the guidelines. First, practitioners need to know which RR approaches are available as tools and which are just algorithms. Knowing this allows for easier selection of tools that can be compared to each other. Such information about empirical comparisons between tools may also be of interest to include in the guidelines. This would require research similar to Li et al.'s work in CIA [1] where he identifies available tools and characterizes them according to the factors he introduces in his framework.

Second, the types of expertise need to be differentiated in further detail as practitioners may be interested in additional kinds of expertise, for example programming language expertise or domain expertise. If this information is acquired and summarized practitioners can much easier identify tools that can determine the kind of expertise they are most interested in.

Thus we see that at least two questions should be added to the guidelines including: i) Which available RR approaches have tool support? and ii) Which tools have been compared to each other? Furthermore, the types of expertise needs to be differentiated in greater detail.

If this research was to be extended another six months, the priority should be to further investigate the additional necessary features required for full adoption of the tool (e.g, including coverage of macros). If these can be implemented, the tool should be more rigorously evaluated by quantitative means such as observational studies or statistical analysis or even experiments. The end goal of such a study would be to quantify the usefulness of CIA tools for improving maintainability. Regarding reviewer recommendation a separate study should be conducted where different reviewer recommendation tools should be implemented and evaluated in practise. The end goal would be to add to the guidelines information about how well different tools perform in different kinds of context to further help practitioners identify tools that may suit their needs.

If the research were to continue for an additional two years, the focus should be to evaluate the effect of using tools for CIA and RR has had on maintainability of testing software as well as continuously revising and adding to the guidelines for choosing RR tools as new insights are found. A key goal that research in both areas should focus on identifying and overcoming the actual challenges hindering transfer of research to industry and between the fields. We found that there are many tools available for both purposes but neither is applied in practise to a significant extent. Furthermore, we believe that tools for CIA may be used to improve the accuracy of RR-tools by providing them with more detailed input information about which files

the RR-tools should calculate expertise for, but we have not found any RR-tools that utilize CIA-tools in this way. Therefore, researchers should apply tools in practise to a larger extent to determine how different approaches perform in different contexts, use tools for CIA and evaluate whether or not such information improves the accuracy of RR-tools and finally evaluate RR-tools usefulness to practitioners.

Lastly, the usefulness of the guidelines should also be more rigorously evaluated by applying them in different contexts to find tools, implement them in practise and evaluate their performance.

Bibliography

[1] Li, B., Sun, X., Leung, H. Zhang, S. 2013, "A survey of code-based change impact analysis techniques", Software Testing, Verification and Reliability, vol. 23, no. 8, pp. 613-646.

[2] Orso, A., Apiwattanapong, T. Harrold, M.J. 2003, "Leveraging field data for impact analysis and regression testing", ACM SIGSOFT Software Engineering Notes, vol. 28, no. 5, pp. 128.

[3] Apiwattanapong, T., Orso, A. Harrold, M. 2005, "Efficient and precise dynamic impact analysis using execute-after sequences", ACM, , pp. 432.

[4] Breech, B., Danalis, A., Shindo, S. Pollock, L. 2004, "Online impact analysis via dynamic compilation technology", IEEE, , pp. 453.

[5] Chen, T.Y., Kuo, F., Liu, H. Wong, W.E. 2013, "Code Coverage of Adaptive Random Testing", IEEE Transactions on Reliability, vol. 62, no. 1, pp. 226-237.

[6] Chen, M.-., Lyu, M.R. Wong, W.E. 2001, "Effect of code coverage on software reliability measurement", IEEE Transactions on Reliability, vol. 50, no. 2, pp. 165-170.

[7] Li, B., Zhang, Q., Sun, X. Leung, H. 2013, "Using water wave propagation phenomenon to study software change impact analysis", Advances in Engineering Software, vol. 58, pp. 45-53.

[8] Acharya, M. Robinson, B. 2011, "Practical change impact analysis based on static program slicing for industrial software systems", ACM, , pp. 746.

[9] Jiang, J., Yang, Y., He, J., Blanc, X. Zhang, L. 2017, "Who should comment on this pull request? Analyzing attributes for more accurate commenter recommendation in pull-based development", Information and Software Technology, vol. 84, pp. 48-62.

[10] Yu, Y., Wang, H., Wang, T. Yin, G. 2016, "Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment?", Information and Software Technology, vol. 74, pp. 204-218.

[11] Prechelt, L., Graziotin, D. Fernández, D.M. 2017, "A Community's Perspective on the Status and Future of Peer Review in Software Engineering",

[12] MacLeod, L., Greiler, M., Storey, M.A., Bird, C. and Czerwonka, J., 2017. Code Reviewing in the Trenches: Understanding Challenges and Best Practices. IEEE Software

[13] Bacchelli, A. Bird, C. 2013, "Expectations, outcomes, and challenges of modern code review", IEEE Press, , pp. 712.

[14] Tao, Y., Dang, Y., Xie, T., Zhang, D. Kim, S. 2012, "How do software engineers understand code changes?: an exploratory study in industry", ACM, , pp. 1.

[15] Thongtanunam, P., Kula, R.G., Cruz, A.E.C., Yoshida, N. Iida, H. 2014, "Improving code review effectiveness through reviewer recommendations", ACM, , pp. 119.

[16] Thongtanunam, P., McIntosh, S., Hassan, A. Iida, H. 2016, "Revisiting code ownership and its relationship with software quality in the scope of modern code review", ACM, , pp. 1039.

[17] Thongtanunam, P., Tantithamthavorn, C., Kula, R.G., Yoshida, N., Iida, H. Matsumoto, K. 2015, "Who should review my code? A file location-based code-reviewer recommendation approach for Modern Code Review", IEEE, , pp. 141.

[18] Szoke, G., Antal, G., Nagy, C., Ferenc, R. Gyimothy, T. 2017;2016;, "Empirical study on refactoring large-scale industrial systems and its effects on maintainability", JOURNAL OF SYSTEMS AND SOFTWARE, vol. 129, pp. 107-126.

[19] Almugrin, S., Albattah, W. Melton, A. 2016, "Using indirect coupling metrics to predict package maintainability and testability", JOURNAL OF SYSTEMS AND SOFTWARE, vol. 121, pp. 298-310.

[20] Midha, V. Bhattacherjee, A. 2012, "Governance practices and software maintenance: A study of open source projects", Decision Support Systems, vol. 54, no. 1, pp. 23-32.

[21] Khan, S., Lee, S., Ahmad, R., Akhunzada, A. Chang, V. 2016, "A survey on Test Suite Reduction frameworks and tools", INTERNATIONAL JOURNAL OF INFORMATION MANAGEMENT, vol. 36, no. 6, pp. 963-975.

[22] Xi, H., 1999, January. Dead code elimination through dependent types. In PADL (Vol. 99, pp. 228-242).

[23] Jiang, S., McMillan, C. Santelices, R. 2017;2016;, "Do Programmers do Change Impact Analysis in Debugging?", EMPIRICAL SOFTWARE ENGINEERING, vol. 22, no. 2, pp. 631-669.

[24] Wang, X., Zhang, Y., Zhao, L. Chen, X. 2017, "Dead Code Detection Method Based on Program Slicing", IEEE, , pp. 155.

[25] Rigby, P.C. Bird, C. 2013, "Convergent contemporary software peer review practices", ACM, , pp. 202.

[26] Faragò, C., Hegeds, P. Ferenc, R. 2015, "Code ownership: Impact on maintainability", , pp. 3.

[27] Nordberg, M.E. 2003, "Managing code ownership", IEEE Software, vol. 20, no. 2, pp. 26-33.

[28] Eldh, S. Murphy, B. 2015, "Code Ownership Perspectives", IEEE Software, vol. 32, no. 6, pp. 18-19.

[29] Knoop, J., Rüthing, O. Steffen, B. 1994, "Partial dead code elimination", ACM, , pp. 147.

[30] Damiani, F. Prost, F. 1998, "Detecting and removing dead-code using rank 2 intersection", , pp. 66.

[31] M. Coppo, F. Damiani, and P. Giannini. Refinement Types for Program Analysis. In SAS'96, LNCS 1145, pages 143–158. Springer, 1996.

[32] Boomsma, H. Gross, H. 2012, "Dead code elimination for web systems written in PHP: Lessons learned from an industry case", IEEE, , pp. 511

[33] S. Berardi. Pruning Simply Typed Lambda Terms. Journal of Symbolic Computation, to appear

[34] [Rafi, D., Moses, K., Petersen, K. Mäntylä, M. 2012, "Benefits and limitations of automated software testing: systematic literature review and practitioner survey", IEEE Press, , pp. 36.]

[35] Wiklund, K., Eldh, S., Sundmark, D. Lundqvist, K. 2017, "Impediments for software test automation: A systematic literature review", Software Testing, Verification and Reliability, vol. 27, no. 8, pp. n/a.

[36] Collins, E., Dias-Neto, A. de Lucena, V.F. 2012, "Strategies for Agile Software Testing Automation: An Industrial Experience", IEEE, , pp. 440.

[37] Wiklund, K., Eldh, S., Sundmark, D. Lundqvist, K. 2012, "Technical Debt in Test Automation", IEEE, , pp. 887.

[38] Leung, H.K.N. White, L. 1991, "A cost model to compare regression test strategies", , pp. 201.

[39] Runeson, P., Ebook Central (e-book collection), Department of Computer Science, Lund University, Institutionen för datavetenskap Lunds universitet 2012, Case study research in software engineering: guidelines and examples, 1. Aufl.;1; edn, Wiley, Hoboken, N.J

[40] Runeson, P., Höst, M., Department of Computer Science, Lund University, Institutionen för datavetenskap Lunds universitet 2009, "Guidelines for conducting and reporting case study research in software engineering", Empirical Software Engineering, vol. 14, no. 2, pp. 131-164.

[41] Hunt, B., Turner, B. McRitchie, K. 2008, "Software Maintenance Implications on Cost and Schedule", IEEE, , pp. 1.

[42] Dehaghani, M.H. Hajrahimi, N. 2013, "Which Factors Affect Software Projects Maintenance Cost More?", Acta Informatica Medica, vol. 21, no. 1, pp. 63.

[43] Zhu, H., Hall, P. May, J. 1997, "Software unit test coverage and adequacy", ACM Computing Surveys (CSUR), vol. 29, no. 4, pp. 366-427.

[44] Kochhar, P.S., Lo, D., Lawall, J. Nagappan, N. 2017, "Code Coverage and Postrelease Defects: A Large-Scale Study on Open Source Projects", IEEE Transactions on Reliability, vol. 66, no. 4, pp. 1213-1228.

[45] Li, J.J., Weiss, D. Yee, H. 2006, "Code-coverage guided prioritized test generation", Information and Software Technology, vol. 48, no. 12, pp. 1187-1198.

[46] Hutchins, M., Foster, H., Goradia, T. Ostrand, T. 1994, "Experiments of the effectiveness of dataflow- and controlflow-based test adequacy criteria", IEEE Computer Society Press, , pp. 191.

[47] Hutchins, M., Foster, H., Goradia, T. Ostrand, T. 1994, "Experiments of the effectiveness of dataflow- and controlflow-based test adequacy criteria", IEEE Computer Society Press, , pp. 191.

[48] Wong, W.E., Horgan, J.R., London, S. Mathur, A.P. 1994, "Effect of test set size and block coverage on the fault detection effectiveness", , pp. 230.

[49] Lin, Y., Chou, C., Lai, Y., Huang, T., Chung, S., Hung, J. Lin, F.C. 2012;2011;, "Test coverage optimization for large code problems", Journal of Systems and Software, vol. 85, no. 1, pp. 16-27.

[50] Godboley, S., Dutta, A., Mohapatra, D. Mall, R. 2018, "GECOJAP: A novel source-code preprocessing technique to improve code coverage", COMPUTER STANDARDS INTERFACES, vol. 55, pp. 27-46.

[51] Perez, A., Abreu, R. Riboira, A. 2014, "A dynamic code coverage approach to maximize fault localization efficiency", JOURNAL OF SYSTEMS AND SOFTWARE, vol. 90, no. 1, pp. 18-28.

[52] Zhang, L., Kim, M. Khurshid, S. 2011, "Localizing failure-inducing program edits based on spectrum information", , pp. 23.

[53] Ren, X., Shah, F., Tip, F., Ryder, B. Chesley, O. 2004, "Chianti: a tool for change impact analysis of java programs", ACM, , pp. 432.

[54] Zimmermann, T., Zeller, A., Weissgerber, P. Diehl, S. 2005, "Mining version histories to guide software changes", IEEE Transactions on Software Engineering, vol. 31, no. 6, pp. 429-445

[55] Poshyvanyk, D. Marcus, A. 2006, "The Conceptual Coupling Metrics for Object-Oriented Systems", IEEE, , pp. 469.

[56] Gethers, M. Poshyvanyk, D. 2010, "Using Relational Topic Models to capture coupling among classes in object-oriented software systems", , pp. 1.

[57] Buckner, J., Buchta, J., Petrenko, M. Rajlich, V. 2005, "JRipples: a tool for program comprehension during incremental change", IEEE, , pp. 149.

[58] Gwizdala, S., Jiang, Y. Rajlich, V. 2003, "Tracker - a tool for change propagation in Java", IEEE, , pp. 223.

[59] Canfora, G. Cerulo, L. 2006, "Jimpa: An Eclipse plug-in for impact analysis", IEEE, , pp. 2 pp.

[60] Bosu, A., Greiler, M. Bird, C. 2015, "Characteristics of useful code reviews: an empirical study at Microsoft", IEEE Press, , pp. 146.

[61] Fu, Q., Grady, F., Broberg, B.F., Roberts, A., Martens, G.G., Johansen, K.V. Loher, P.L. 2017, "Code review and cooperative pair programming best practice",

[62] Ouni, A., Kula, R.G. Inoue, K. 2016, "Search-Based Peer Reviewers Recommendation in Modern Code Review", IEEE, , pp. 367.

[63] Zanjani, M.B., Kagdi, H. Bird, C. 2016, "Automatically Recommending Peer Reviewers in Modern Code Review", IEEE Transactions on Software Engineering, vol. 42, no. 6, pp. 530-543.

[64] Kagdi, H., Hammad, M. Maletic, J.I. 2008, "Who can help me with this source code change?", IEEE, , pp. 157.

[65] Hannebauer, C., Patalas, M., Stünkel, S. Gruhn, V. 2016, "Automatically recommending code reviewers based on their expertise: an empirical comparison", ACM, , pp. 99.

[66] McDonald, D. Ackerman, M. 2000, "Expertise recommender: a flexible recommendation system and architecture", ACM, , pp. 231.

[67] Xia, Z., Sun, H., Jiang, J., Wang, X. Liu, X. 2017, "A hybrid approach to code reviewer recommendation with collaborative filtering", IEEE, , pp. 24.

[68] Xia, X., Lo, D., Wang, X. Yang, X. 2015, "Who should review this change?: Putting text and file location analyses together for more accurate recommendations", IEEE, , pp. 261.

[69] Balachandran, V. 2013, "Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation", IEEE Press, , pp. 931.

[70] Alonso, O., Devanbu, P. Gertz, M. 2008, "Expertise identification and visualization from CVS", ACM, , pp. 125.

[71] Girba, T., Kuhn, A., Seeberger, M. Ducasse, S. 2005, "How developers drive software evolution", IEEE, , pp. 113.

[72] Bird, C., Nagappan, N., Murphy, B., Gall, H. Devanbu, P. 2011, "Don't touch my code: examining the effects of ownership on software quality", ACM, , pp. 4.

[73] Pfleeger, S. Kitchenham, B. 2001, "Principles of survey research: part 1: turning lemons into lemonade", ACM SIGSOFT Software Engineering Notes, vol. 26, no. 6, pp. 16-18.

[74] Kitchenham, B. Pfleeger, S. 2002, "Principles of survey research part 2: designing a survey", ACM SIGSOFT Software Engineering Notes, vol. 27, no. 1, pp. 18-20.

[75] Kitchenham, B. Pfleeger, S. 2002, "Principles of survey research: part 3: constructing a survey instrument", ACM SIGSOFT Software Engineering Notes, vol. 27, no. 2, pp. 20-24.

[76] Kitchenham, B. Pfleeger, S. 2002, "Principles of survey research part 4: questionnaire evaluation", ACM SIGSOFT Software Engineering Notes, vol. 27, no. 3, pp. 20-23.

[77] Kitchenham, B. Pfleeger, S. 2002, "Principles of survey research: part 5: populations and samples", ACM SIGSOFT Software Engineering Notes, vol. 27, no. 5, pp. 17-20.

[78] Kitchenham, B. Pfleeger, S. 2003, "Principles of survey research part 6: data analysis", ACM SIGSOFT Software Engineering Notes, vol. 28, no. 2, pp. 24-27.

[79] Fritz, T., Murphy, G.C., Murphy-Hill, E., Ou, J. Hill, E. 2014, "Degree-of-knowledge: Modeling a developer's knowledge of code", ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 23, no. 2, pp. 1-42.

[80] Rapos, E.J. Cordy, J.R. 2017, "SimPact: Impact Analysis for Simulink Models", IEEE, , pp. 489.

[81] Rapos, E.J. Cordy, J.R. 2017, "SimEvo: A Toolset for Simulink Test Evolution Maintenance", IEEE, , .

[82] IEEE Xplore Standards (e-book collection) IEEE Xplore (e-book collection) 2008, IEEE Std 1028-2008: IEEE Standard for Software Reviews and Audits, IEEE, S.l.

[83] IEEE Xplore Standards (e-book collection) IEEE Xplore (e-book collection) IEEE Standard for Software Maintenance, IEEE, Piscataway.

[84] IEEE Xplore Standards (e-book collection) IEEE Xplore (e-book collection) 2013, ISO/IEC/IEEE 29119-1: 2013(E): Software and systems engineering Software testing Part 1:Concepts and definitions, IEEE, S.l.

[85] Sae-Lim, N., Hayashi, S. Saeki, M. 2016, "Context-based code smells prioritization for prefactoring", IEEE, , pp. 1.

[86] Kasurinen, J., Taipale, O. Smolander, K. 2010, "Software Test Automation in Practice: Empirical Observations", Advances in Software Engineering, vol. 2010, pp. 1-18.

[87] Berner, S., Weber, R. Keller, R. 2005, "Observations and lessons learned from automated testing", ACM, , pp. 571.

[88] Karlström, D., Runeson, P., Nordén, S., Department of Computer Science, Lund University, Institutionen för datavetenskap Lunds universitet 2005, "A minimal test practice framework for emerging software organizations", Software Testing, Verification and Reliability, vol. 15, no. 3, pp. 145-166.

[89] N. Kerzazi, F. Khomh, and B. Adams, “Why do automated builds break? an empirical study,” in 2014 IEEE International Conference on Software Maintenance and Evolution, Sept 2014, pp. 41–50

[90] McKee, S., Nelson, N., Sarma, A. Dig, D. 2017, "Software Practitioner Perspectives on Merge Conflicts and Resolutions", IEEE, , pp. 467.

[91] Yoo, S. Harman, M. 2010;2012;, "Regression testing minimization, selection and prioritization: a survey", Software Testing, Verification and Reliability, vol. 22, no. 2, pp. 67-n/a.

A

Appendix 1

A.1 Interview instrument for initial interviews

Theme	Question	Purpose
General working process	1) What is your current process? a) Flow of your process step by step b) Do you make changes in the framework? c) If so when?	Understanding how the interviewee works and their role in the framework.
Understanding code	2) How do you go about understanding code? a) How do you find knowledgeable developers to help you get started/explain code for you? b) What tools do you use for this? c) Do you use documentation to understand a code and if so how?	Understanding how maintainers understand code.
Finding reviewers	**XFT and LSV** 3) How do you find reviewers for your code? a) Who are you usually looking for? b) What tools do you have available? c) What channels of communication do you have available to find reviewers? d) How fast can you find a reviewer? e) How fast do you usually receive a response? f) How often can you find a suitable reviewer? **TOR** 3) How are you approached by with review requests? a) How often do you receive requests? b) How often are you requested for the code you are not expert on? c) How do you handle such requests? d) How do you find appropriate people when assigning review requests?	Understanding how maintainers find reviewers.
Coding and code review practises	4) What are your practices? a) How often do you commit? b) How big/complex are your commits? c) How do you document your work?	Understanding the practises of the maintainer.
Change impact analysis	5) Do you perform CIA and if so how? a) What tools are used? b) How often and to what extent? c) Who is involved? d) How do you document your changes?	Understanding the role of CIA and how it is performed at Ericsson.

Figure A.1: The interview instrument for the initial interviews

A.2 Interview instrument for the CIA tool

Theme	Question	Purpose
General use of tool	1) How did you use the tool? (a) Was it helpful in any way and if so how?	To find out how the interviewees used the tool and for what purposes.
General use of tool	2) Did you gain any insights from using the tool? (a) If so what were they? (b) What did you use these insight for? (c) What insights did you expect/want from the tool?	To find out what the interviewees learned by using the tool
Understandability	3) Did the information provided from the tool help you understand the code you were working with? (a) If so, which elements of the code did the information help you understand? (b) In what way did the information help you understand the code?	To find out if the tool was helpful in improving understandability of the TF
Modifiability	4) Did the information provided from the tool help you in determining the impact of your modifications to the code? (a) If so, in what ways was the information helpful?	To find out if the tool was helpful in improving modifiability of the TF
Future extensions/modifications	5) Is there anything you would want to change with the tool to make it more useful? (a) Is there anything you would modify with the tool? (b) Is there anything you would want to add to the tool? (c) is there anything you would remove from the tool?	To find out possible flaws and improvements of the tool.

Figure A.2: The interview instrument for the first set of interviews

A.3 Interview instrument for the guidelines

The interview instrument we used to evaluate the usefulness of the recommendation for how to facilitate finding reviewers are shown below:

Theme	Question	Purpose
Novelty of information and general thoughts	1) What are your thoughts about the guidelines? i) Did the guidelines provide you with any new insights? ii) If so what were they?	Eliciting initial thoughts and finding out if the guidelines provide new information about RR tools.
Application of guidelines	2) What are your thought about how the guidelines were applied at Ericsson? i) Do you agree with the results? ii) Is there anything that you consider relevant to the conclusions that was not considered?	Verifying that the application of the guidelines was done correctly and not overlooking key factors.
Results	3) Would you choose the same tool? i) Why/why not?	Finding out what is most important for Ericsson regarding an RR tool
Usefulness of guidelines	4) What do you see as a next step for Ericsson regarding ARR tools? i) Have the guidelines been useful for furthering your understanding of RR tools and if so how?	Evaluating the usefulness of the guidelines.

Figure A.3: The interview instrument for the first set of interviews

A.4 List of investigated RR approaches

We investigated several tools that turned out to be inferior to at least one of the three provided in table 1. Below we present the tools that each tool in table 1 were proven to outperform.

WRC proven better than:

1. FPS (achieves similar results to FPS but is simpler and much faster) [65, 15]
2. Line 10 Rule [65]
3. Expertise Recommender [66]
4. Number of Changes [65]
5. Code Ownership [71]
6. Expertise Cloud [70]
7. Degree-of-Authorship [79]

RevRec proven better than:

1. cHRev [63]. Which in turn outperforms:
   (a) xFinder [64]
   (b) Revcom [63]
   (c) REVFINDER
2. REVFINDER [15]
3. ReviewBot [69].

PR-CF proven better than:

1. IR+CN[10]
2. FPS [65, 15]
3. activeness [23]
4. TIE [68]

B

Appendix 2

Coverage tool Training

Coverage data shows which parts of the application code are executed (covered by the execution) and which parts are not executed. Here, the application code is the code in **`/epg/epgcats/tcdb/*`**

Every test execution will generate coverage info which is stored in the autoTT log directory htmlcov. The html report of the coverage data is also stored in the log directory, which can be used as a graphical tool to see which parts of the application are executed and which parts are not executed for a specific test case.

For weekend release candidate builds we can collect the coverage data for all autott test executions started by the portal test scheduler (e.g by user stportal) using the tools described below. With this data we can find out which testcases hit which lines of code.

Tools location:

The tools are found in **epg/epgcats/tools/coverage**

collecting the coverage data of all test runs

Autott's weekly execution will generate coverage data for each test cases, to collect all the coverage data from each test case, use the following command.

./create_coverage <BUILD>

This command will collect all the coverage data from all test cases under that build and store it in a json file.

<BUILD> has the following format ***EPG_<NUMBER>R<NUMBER><A-Z><NUMBER>_<DATE>_<TIMESTAMP>*** example: ***EPG_25R12A459_180420_082553***

Example: ./create_coverage EPG_25R12A459_180420_082553

As can be seen from the figure, in the build (*EPG_25R12A459_180420_082553*) 6 test cases are found. The '.json' file is created if the `create_coverage` is run for the first time for that specific build. If it is run more than once for the same build, the .json file will be recreated with the additional data.

Checking coverage info

Coverage information is collected once a week so the latest data is from last weeks run. To check which test case run has executed(covered) a specific part of a file, you can specify the file name and the line number and get the specific coverage data for that specific build.

./cia <file_name> <line_number> -b <BUILD>

file_name is the name of the python file which you want to check (the name includes the path starting after tcdb) Example: lib/general/yaml_emitter.py or lib_general_yaml_emitter.py

line_number is the line number in the file which you want to check.

Build is the build name (optional field), if no Build is provided it shows the test cases which covered this file name with line number line_number in every build where data has been collected.

NOTE: For a filename, the relative path starts after epg/epgcats/tcdb/ E.g. if the file is epg/epgcats/tcdb/LIB_general/tunelib.py : your relative path will be LIB_general/tunelib.py

Example 1 : ./cia lib_general_robustness_ext_rob.py 374

```
(coverage)> ./cia lib_general_robustness_ext_rob.py 374

['TC2.1_ssr_ssc1_cp_ssc3_up', 'TC15934', 'TC2.1_ssr_sr', 'TC15934.25', 'TC16938.41', 'TC00_cot
_cp_ssc3_up', 'TC2.1_cots_CMCC_DirectIO'] in EPG_25R12A363_180413_130211

['TC00_cots_1host'] in EPG_25R12A456_180420_060546

['TC2.1_cots_1host', 'TC00_cots_1host', 'TC19881', 'TC8981.8.2', 'TC15934.5', 'TC2.1_ssr_ssc1_
ssc3_cp_ssc3_up'] in EPG_25R12A459_180420_082553



              Detailed HTML Report can be found in the below links


TC2.1_ssr_ssc1_cp_ssc3_up in EPG_25R12A363_180413_130211
/lab/epg_st_portal_logs/EPG_25R12A363_180413_130211_5487905_stportal/2018-04-13_20.26_TC2.1_ss
cp_ssc3_up_robustness_ssr8020s2/htmlcov/_lab_epg_scm4_builds_program_ci_EPG_25R12A363_epgcats_

TC2.1_ssr_ssc1_cp_ssc3_up in EPG_25R12A363_180413_130211
/lab/epg_st_portal_logs/EPG_25R12A363_180413_130211_5487985_stportal/2018-04-14_12.58_TC2.1_ss
020s2/htmlcov/_lab_epg_scm4_builds_program_ci_EPG_25R12A363_epgcats_tcdb_LIB_general_Robustnes
```

As can be seen from the figure, if the file name and the line number of the file is given, test cases that cover this line number of the file will be printed. In the figure above, *ext_rob.py* file at line 374 is executed by 7 test cases in the first build and by one test case in the second build and by 6 test cases in the last build(If you want to get the result for a specific build, then pass the build name as shown in Example 2). A detailed description of the coverage of the file for a specific build can be found in the html report (the listed link).

the link can be opened as (p*lease use the full path), and the report looks like the figure below*

xdg-open ....._lab_epg_scm4_builds_program_ci_EPG_25R12A309_epgcats_tcdb_LIB_general_robustness_ext_rob.py.html

```python
#! C:/python.exe
import os
import subprocess
import coverage
import sys

os.environ['COVERAGE_PROCESS_START'] = 'C:\Lib\site-packages\.coveragerc'


def test_method():
    sum = 0
    for i in range(100):
        sum +=i
    print("dummy method output is : {}".format(sum))
    call_python2()


def call_python2():
    subprocess.call(os.path.join(os.path.dirname(os.path.realpath(__file__)), 'python2_script.py'), shell=True)


if __name__ == '__main__':
    print("python3 - v {}".format(sys.version))
    cov = coverage.coverage()
    cov.start()
    test_method()
    cov.stop()
    cov.combine()
    cov.html_report()
    cov.save()
```

N.B *The Green part shows the part that is executed and the red part shows the part which isn't executed.*

Example 2: ***`./cia lib_general_robustness_ext_rob.py 374 -b EPG_25R12A459_180420_082553`***

```
          (coverage)> ./cia lib_general_robustness_ext_rob.py 374 -b EPG_25R12A459_180420_0825
['TC2.1_cots_1host', 'TC00_cots_1host', 'TC19881', 'TC8981.8.2', 'TC15934.5', 'TC2.1_ssr_ssc1_c
ssc3_cp_ssc3_up']



              Detailed HTML Report can be found in the below links


TC2.1_cots_1host in EPG_25R12A459_180420_082553
/lab/epg_st_portal_logs/EPG_25R12A459_180420_082553_5551736_stportal/2018-04-21_19.00_TC2.1_cot
-vIP-Ps2/htmlcov/_lab_epg_scm4_builds_program_ci_EPG_25R12A459_epgcats_tcdb_LIB_general_Robustr

TC2.1_cots_1host in EPG_25R12A459_180420_082553
/lab/epg_st_portal_logs/EPG_25R12A459_180420_082553_5547711_stportal/2018-04-20_12.08_TC2.1_cot
-vIP-Ps2/htmlcov/_lab_epg_scm4_builds_program_ci_EPG_25R12A459_epgcats_tcdb_LIB_general_Robustr
```

This Example shows how to get the coverage data for a specific build.

N.B There is a debugger option which is configured to **logging.DEBUG,** but if you wish to have no DEBUG info, then you can remove this by passing **-debug info** option

Notes:

1. **As you know, the line numbers change when files are edited. So if you are looking at a source code that has changed since the last coverage run, you need to look at the file vesion that was valid for that build. The HTML REPORT is a good input on those**

cases, since it will show you the executed source code itself.

2. **Coverage data is only fetched for “stportal” executions.This means re-runs are not covered yet. (reruns are usually not run by stportal user).**
3. **Dont choose line numbers where the function is defined e.g *def myfunc* since any time a python file is imported by another coverage marks the line as being covered. Instead use the first line of code in the function if you want to see where the function is used.**

Coverage of other code

autoTT provides options to execute coverage on more code than what is included in LIB_general.

To add more use the autoTT option -o cov=path1,path2