CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Evaluation of Text–Based Requirements Engineering Tools

An in-depth case study at Ericsson

Master's thesis in Computer science and engineering

Auður Katarína Theodórsdóttir

Nora Ojensa

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

Master's thesis 2023

Evaluation of Text–Based Requirements Engineering Tools

An in-depth case study at Ericsson

Auður Katarína Theodórsdóttir
Nora Ojensa

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

Evaluation of Text–Based Requirements Engineering Tools
An in-depth case study at Ericsson
Auður Katarína Theodórsdóttir & Nora Ojensa

Supervisor: Eric Knauss, Department of Computer Science and Engineering
Advisor: Filip Lange, Ericsson
Examiner: Gregory Gay, Department of Computer Science and Engineering

Master's Thesis 2023
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2023

Evaluation of Text–Based Requirements Engineering Tools
An in-depth case study at Ericsson
Auður Katarína Theodórsdóttir & Nora Ojensa
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Context: Additional challenges related to requirements engineering practices arise as software development moves from the waterfall model towards agile development, where continuous integration and continuous deployment are often employed. This change has resulted in the emergence of new requirements engineering tools. One of which is a tool developed in-house at Ericsson by the name of T-Reqs.
Objective: The purpose of this study is to study these newly emerged tools by conducting a case study at Ericsson to evaluate T-Reqs, i.e., illustrate its advantages and disadvantages, as well as display how the tool is used in practice in comparison to traditional requirements engineering tools. In addition, an identification of the tool's characteristics that also describe tools similar to T-Reqs and a formal definition for these tools is proposed.
Method: The chosen research method is an in-depth case study where the data collection was both qualitative and quantitative, i.e., a mixed method design. The qualitative data was gathered through 14 semi-structured interviews with relevant employees at Ericsson and followed by quantitative data gathering through a follow-up survey. The results were then discussed in a cross-industry workshop.
Results: The results illustrate that the primary expectations towards T-Reqs were increased usability and improved ways of working, which encompassed a scale-out of the company and facilitated collaboration, among others. These expectations were, for the most part, fulfilled. One of the main advantages identified was the tool's integration into the development environment, allowing for easy access, updates, version control, and baselining. As the tool relies heavily on its users being comfortable with using Git version control, one of the main drawbacks of the tool relates to its ease of use for users who lack the knowledge of standard IT technology. Finally, the main characteristics of the tool are that it is connected to version control, integrated into each product's repository, and its ease of use. From these characteristics, a formal definition of these newly emerged tools is suggested, referred to as *text-based requirements management tools*.
Conclusion: This study demonstrates that T-Reqs has been successfully integrated into the development processes at Ericsson, providing many advantages over traditional requirements engineering tools. Thus, T-Reqs can be offered as an example of how these text-based requirements management tools can facilitate the development process in large-scale agile environments, once their implementation is carefully tailored to the unique needs and context of the organization.

Keywords: requirements engineering, large-scale agile development, agile requirements engineering, requirements engineering tools, text-based requirements management tools

Acknowledgements

We want to express our gratitude to our supervisor, Eric Knauss, for his invaluable support and guidance. We further thank Filip Lange for his contribution and help during this thesis work as well as the enjoyable collaboration during the course Industrial Practice Project which led to this thesis. In addition, we want to thank our examiner Gregory Gay for his helpful comments and all the participants of the interviews and surveys for taking the time to answer our questions and offer their insights. Furthermore, we are thankful to the workshop participants at the Software Center for sharing their perspectives and to Eric for inviting us to present our thesis project. Finally, we want to thank our families for their continued love and support, for which we are eternally grateful.

Auður Katarína Theodórsdóttir & Nora Ojensa
Gothenburg, June 2023

Contents

1
Introduction

Due to software engineering practices moving toward continuous integration (CI) and continuous deployment (CD), the more traditional ways of gathering and formulating requirements to support a waterfall operation are less feasible. This is due to the flexible environment CI/CD and an agile workflow present. To allocate a more extended time to formulate requirements that further on are likely to change can be a waste of valuable time and money. Thus, when introducing these agile methods, typically less time is allocated to the construction of requirements. Spending less time on the requirements often results in developers focusing solely on functional requirements described in the form of user stories [1] and losing the level of detail associated with requirements gathered through more plan-driven approaches [2]. In abandoning the more plan-driven approaches, agile methods introduce a greater focus on close collaboration with stakeholders and customers, and thus requirements must be available for changes during the development process. Using agile methods to perform requirements engineering is often referred to as agile requirements engineering. However, a challenge presents itself when using this terminology as, according to Heikkilä et al. [3], agile requirements engineering does not have a universal definition.

To address these challenges, new requirements engineering tools have emerged that support an agile environment, which is very susceptible to change. They rely on Git-based knowledge storage, i.e., a version control system, which provides the opportunity to connect requirements more closely to the code base as well as the test suite of a system. In doing so, these tools also bring requirements closer to the developers. Among these tools, the most notable are Doorstop [4] and T-Reqs [7] (refer to T-Reqs's tool description in Section 3.2). The latter has both an in-house produced version at Ericsson and an open-source version developed by Chalmers University of Technology and Gothenburg University [6]. Ericsson is a telecommunications company based in Sweden, where large-scale agile systems development is employed.

Five years have elapsed since Ericsson adopted T-Reqs into their requirements engineering processes. In a previous study [7], the challenges mitigated by T-Reqs have been showcased. However, the study did not encompass the evaluation of how T-Reqs is used in practice and if these mitigations are fully utilized by users. Furthermore, the question of whether Ericsson's expectations for T-Reqs, when it was implemented, have been met still needs to be answered. The study aims to answer these questions by conducting an in-depth case study at Ericsson and collecting qualitative and quantitative data through semi-structured interviews [37] with employees at Ericsson and a follow-up survey. The employees who were interviewed

are the current users of T-Reqs, i.e., testers and system managers. The objective was to identify and highlight its successful aspects while also examining potential drawbacks. This analysis provides insights that can be contextualized within the realm of similar requirements engineering tools relying on a version control system, e.g., Git. In general, it can showcase themes that can be valuable to companies with similar environments, using Git or any version control system in their requirements engineering tool. Further, in order to increase the generalizability and transferability of the results, an additional data collection method was conducted in the form of a workshop with experts in the industry on the topics of requirements and tooling.

1.1 Statement of the Problem

Gathering and formulating requirements by following the waterfall model has proven to not be feasible in an agile environment utilizing CI/CD. Requirements have to be gathered fast and must be susceptible to change during the development process [8]. This has resulted in new requirements engineering tools being on the rise that support agile work. Since these requirements engineering tools are quite new in the industry, there is not a lot of research on them in general, let alone on their performance. The problem is, therefore, that it is not entirely clear how these tools can be classified and how well they are performing in the industry. That is, questions arise as to whether these tools are meeting the desired expectations, i.e., solving the challenges faced by using the more traditional tools, in addition to whether there are perhaps some unexpected benefits or drawbacks from using these tools.

1.2 Purpose of the Study

The purpose of this study is to get a better understanding of the new requirements engineering tools that have surfaced to mitigate the challenges that have arisen due to employing requirements engineering in this different development environment. That is, how are they used in practice, their observed advantages and disadvantages, and moreover, their characteristics. In order to increase our knowledge of these tools, an in-depth case study was performed to evaluate the text-based requirements engineering tool T-Reqs, currently being used at Ericsson.

We aim to evaluate the current state of the tool, how it is used, and how the tool performs in practice. This is mainly to systematically identify and assess the advantages and disadvantages of T-Reqs, additionally uncovering unknown challenges possibly being introduced with T-Reqs. The assessment will focus on determining whether the tool meets its expectations and successfully addresses the challenges faced by Ericsson when they used traditional requirements engineering tools such as IBM Rational RequisitePro and Rational Rhapsody. By doing so, the study aims to determine if T-Reqs effectively facilitates the requirements engineering process at Ericsson.

The result will be used to characterize T-Reqs and tools with similar characteristics and propose a formal definition for them. This study aims to illustrate as well the advantages and disadvantages of using said tools, and the results can

benefit Ericsson as well as other companies operating in a similar company context, i.e., large-scale agile development. Based on the results of this study, companies can identify whether tools similar to T-Reqs would be a good alternative for their development processes. Since the study is structured as a case study, as remarked by Stol et al. [43], generalization is not the goal. However, by displaying the context in which the tool is utilized and the expressed experiences of requirements engineering practitioners gathered from a workshop, transferability, and generalization of the findings can be applied to a certain extent. Further, by illustrating the characterization of T-Reqs and constructing a proposal for a definition of this class of requirements engineering tools, we contribute to the research knowledge and suggest directions for future research.

1.3 Research Questions

To evaluate the text-based requirements engineering tool T-Reqs, we formulated the following research questions:

> **RQ1**: How well do the expectations of T-Reqs before its implementation match with how the tool performs today?
>
> **RQ2**: What advantages and disadvantages of T-Reqs can be observed in practice in relation to traditional requirement tools?
>
> **RQ3**: What characterizes tools such as T-Reqs in contrast with more traditional requirements tools?

These research questions help to illuminate the value that T-Reqs brings to the requirements engineering process at Ericsson in comparison to traditional requirements engineering tools. This, consequently, will showcase to companies following similar development practices if tools such as T-Reqs would be a suitable option to manage their requirements.

1.4 Significance of the Study

By conducting an in-depth case study of practices and the requirements engineering tool at Ericsson, we are able to contribute qualitative results to the research community in the area of requirements engineering in large-scale agile system development. The research builds upon the information regarding the exploration of alternatives to traditional requirements engineering tools, which is beneficial for both practitioners and researchers. That is, it is beneficial for practitioners in the sense that companies will be able to make better decisions regarding their requirements engineering practices and which tools are most suitable for them. It benefits the research community as the study also questions current research, and the results are valuable as to know in which areas to focus on when discussing requirements engineering tools in large-scale agile environments. A formal definition of the group of tools T-Reqs

belongs to will facilitate the discussion regarding requirements engineering tools in a scientific manner.

1.5 Outline

This thesis report is organized as follows. In Section 2, the topics related to this study will be explained by illustrating the prior related work that has been performed in those areas. In Section 3, the case context will be explained by mentioning the previous relevant research performed at Ericsson and what each role at the company encompasses. In Section 4, the methodology of the study is explained, as well as the different data collection methods and analysis. The threats to the validity of the study will also be discussed. In Section 5, the results from the semi-structured interviews, the survey, and the workshop will be reported in accordance with the research questions posed in this study. In Section 6, the results in the previous section will be reflected upon and discussed, and the conclusion of the report is in Section 7.

2
Background and Related Work

The topics of this research, i.e., requirements engineering, agile development, and agile requirements engineering, will be discussed in this chapter in relation to previous research on these topics. Furthermore, requirements engineering and management tools will be discussed along with different evaluation methods previously employed.

2.1 Requirements Engineering

According to the International Requirements Engineering Board, IREB, requirements engineering is defined as "The systematic and disciplined approach to the specification and management of requirements with the goal of understanding the stakeholders' desires and needs and minimizing the risk of delivering a system that does not meet these desires and needs." (p. 11) [22]. Thus, it is important to note that requirements engineering is a multifaceted practice, that is, not only defining the requirements and constraints of a system but the entire process of managing them. IREB specifies that the main activities in performing requirements engineering include elicitation, documentation, validation, and management tasks, which also cover requirements analysis and conflict resolution. [22]. In addition to the previously mentioned tasks, requirements verification, requirements reuse, and requirements quality assessment are also noted in the ISO standard ISO/IEC/IEEE 29148:2018 [23]. The standard illustrates the processes and products when performing requirements engineering in the life cycle of the system and software.

In contrast to the IREB classification, the standard does not declare documentation as a specific task but rather mentions different types of requirements specifications. Furthermore, requirements analysis is also treated as a separate task instead of being classified as part of elicitation. This classification is further illustrated in the ISO standard ISO/IEC TR 24766:2009 [24], which is a guide for the expected capabilities of requirements engineering tools. It lists 158 requirements engineering tool capabilities which pertain to six different requirements engineering tasks. This standard is discussed in more detail in Section 2.4. In this report, we will mainly concern ourselves with the requirements engineering tasks that T-Reqs encompasses: documentation (including specification), validation and verification, and requirements management, i.e., change management and traceability.

A study by The Standish Group, where IT executive managers were surveyed and interviewed, indicates that the practice of requirements engineering is essential for a successful software project [20]. The study included 375 respondents from companies from various industries and sizes. The results show that incomplete

requirements are one of the main reasons why software projects fail, while clearly stated requirements are one of the major contributing factors to a successful project. Requirements engineering is also stated as the most important activity in the development of complex and software-intensive systems, according to Konrad et al. [21]. As the complexity of a system under development increases, so does the importance of good requirements engineering.

In spite of the importance of requirements engineering, there is no universal process that states how requirements engineering should be conducted or when in the development process of a system. The most suitable requirements engineering process must be chosen each time depending, for example, on the development context or the overall development process [22].

2.2 Agile Development

Agile software development officially dates back to 2001 when the "Agile Manifesto" was constructed [2]. It emerged as a response to the previous development method, i.e., plan-driven software development. Plan-driven software development, also termed traditional software engineering, is an up-front development process [2, 17, 18]. Well-defined iterations, extensive documentation, and an exhaustive requirement design stage at the start of the project are some of the core principles. In contrast, agile development aims to mitigate some initial up-front documentation and be more flexible in managing change and getting products to market faster. This includes designing requirements, which would be done throughout the development process.

2.2.1 Large-Scale Agile Development

Introducing agile development practices to companies with multiple development teams and larger projects can pose a challenge as these practices relate more to individual teams, such as extreme programming, XP [38–40], and do not thus account for coordination between different teams or different parts of the organization. To account for these challenges, agile development practices have been expanded into different frameworks, e.g., Scaled Agile Framework (SAFe) and Large-Scale Scrum (LeSS).

In a systematic literature review by Dikert et al. [39], 42 cases of adoption of agile at scale from the industry were examined. A company was considered large-scale if it had 50 employees or more or a minimum of six teams. The reported challenges identified were 35 and 29 success factors. The challenges were grouped into nine categories, with the highest percentage of occurrences in the cases being: agile difficult to implement, integrating non-development functions, requirements engineering challenges, and change resistance. The success factors were grouped into eleven categories, e.g., management support, commitment to change, choosing and customizing the agile approach, and requirements management. From these challenges and success factors, it is evident that there must be a desire in the company for a change, both high and low in the company's hierarchy, and important to

find a good solution that will fit the company context. Requirements also play a big role, and the importance of requirements management is emphasized.

In a multiple case study by Kasauli et al. [41], challenges and practices regarding requirements engineering at seven large-scale system companies were identified. The data was collected and validated through 20 interviews, five focus groups, and eight workshops. From the data, 24 challenges were identified and grouped into six themes, i.e., (C1) build and maintain shared understanding of customer value, (C2) support change and evolution, (C3) build and maintain shared understanding about system, (C4) representation of requirements knowledge, (C5) process aspects and (C6) organizational aspects. In the study, T-Reqs is mentioned as a solution for some of these challenges. More specifically, the challenges of managing experimental requirements and updating requirements, i.e., when and who should update the requirements, under C2. Under C4, the challenge of consistent requirements quality across teams and boundaries. Finally, under C6, the challenge of bridging plan-driven and agile, i.e., between system level and team level.

2.3 Agile Requirements Engineering

A systematic literature review on agile requirements engineering practices and challenges was performed by Inayat et al. [14]. Literature published between 2002 and June 2013 was reviewed, which resulted in 21 papers being identified that discussed agile requirements engineering. Among these papers was a case study by Bjarnason et al. [15] identifying benefits and side-effects of agile practices in large-scale requirements engineering and an empirical study conducted by Ramesh et al. [16] relating agile requirements engineering practices and challenges. The review identified 17 practices relating to agile requirements engineering, and eight challenges posed by those practices. Additionally, it made note of five challenges stemming from traditional requirements engineering, which were mitigated by agile requirements engineering.

The 17 requirements engineering practices found to be adopted in agile software development are, for example, face-to-face communication, iterative requirements, change management, cross-functional teams, and requirements management. According to Ramesh et al., "Accommodating requirements changes during development is a way of tuning the system to better satisfy customer needs" (p. 65) [16]. They further noted that the most requested changes to the requirements were to either add or drop features.

The five challenges linked to traditional requirements engineering that are solved by agile requirements engineering are, namely, communication issues, overscoping, requirements validation, requirements documentation, and rare customer involvement. As stated by Bjarnason et al. [15], communication gaps are solved, for example, by having cross-functional teams, gradual detailing of requirements, and the practice of having an integrated requirements engineering process. The integration entails bringing the requirement engineering process closer to the development in the hope of bettering the developers' understanding and improving communication. As mentioned above, this is an expected outcome of using T-Reqs, i.e., bringing requirements closer to the developers. Tools such as T-Reqs are expected

to align with the practices of agile requirements engineering and facilitate the process. Thus, by illustrating the findings of the literature above, the expectation is that this corresponds with the experience of the employees at Ericsson.

It is important to note that, as mentioned above, agile requirements engineering does not have a universal definition as stated by Heikkilä et al. [3] and thus presents an obvious challenge when discussing the topic. In the same paper, they note that the practices of agile and traditional requirements engineering have similar goals. However, the difference lies in the methods and emphases, where on the one hand, traditional requirements engineering emphasizes processes and documents. On the other hand, agile requirements engineering emphasizes reactivity and informal communication.

2.4 Requirements Tools

Tools exist to aid developers in the software development process. These tools have been referred to as CASE, i.e., computer-aided software engineering tools [29]. Among those tools are tools relating to requirements engineering. Both IREB and ISO/IEC TR 24766:2009 state the importance of using requirements engineering tools to perform requirements engineering tasks whenever possible [22, 24]. The standard ISO/IEC/IEEE 29148:2018 attests to this as well, particularly to employ requirements management tools in more complex projects [23]. ArgonDigital (previously Seilevel) also emphasizes this by stating that it is critical that a tool for requirements management is adopted, especially in larger corporations where there can be hundreds of thousands of requirements for a project [28].

The terminology used to describe requirements engineering tools varies across different studies, with the terms "requirements engineering tools" and "requirements management tools" often used interchangeably. In this paper, we will regard requirements engineering tools as tools supporting the entire requirements engineering process and all its activities, while requirements management tools facilitate the management of requirements once they are created, as stated in the standard ISO/IEC TR 24766:2009 [24]. To further illustrate this, the definition given in the CPRE glossary published by IREB states that requirements management is "The process of managing existing requirements and requirements related work products, including the storing, changing and tracing of requirements." (p. 17) [25].

Available requirements engineering tools vary in functionality and the requirements engineering activities that they support. There has also been a rapidly changing market as many new tools have emerged and old ones have become deprecated, with feature sets changing widely, as noted by ArgonDigital [28], and Gea et al. [34]. Where the newly emerging tools increasingly support visual modeling, traceability analysis, and collaborative functionality for teams. ArgonDigital has made evaluation reports comparing many tools in the industry, published in the years 2007, 2011, and 2016, where in their last report, they compared 175 different requirements engineering tools. Selecting an inappropriate tool or one with excessive complexity can impede the progress of a project, as noted by Young [33]. He emphasizes that a requirements engineering tool should not be chosen out of convenience or arbitrary criteria but the need to consider what is best for the operations.

With new tools emerging with new feature sets, the market is not homogeneous, and groupings should be represented in the terminology. A recent study presenting a newer tool, Doorstop, has used the terminology of centralized, cloud-based, and decentralized to categorize requirements management tools. Centralized tools are a more traditional way of creating and storing requirements by using a centralized hosted database for the storage of requirements and a dedicated program with a graphical user interface. Cloud-based tools offer a web browser for their users to interact with to collaborate on their requirements documentation. Lastly, decentralized tools store requirements as files instead of database entries, thus eliminating the requirement for a dedicated server. Decentralized tools, T-Reqs therein, have changed the way that requirements management can be conducted by connecting requirements and the source code closer together [4].

Another way of categorizing these requirements tools is based on which requirements engineering tasks they cover. As an example, IREB states in their handbook that differentiating between different tools can be done based on the following requirements engineering aspects, i.e., management of requirements, requirements engineering process, documentation of the knowledge about the requirements, modeling of requirements, collaboration in requirements engineering, and testing and/or simulation of the requirements. They further state that most tools do not cover all of these tasks, and thus a combination of different tools must be considered to fully support the whole requirements engineering process [22]. Similarly, the ISO standard ISO/IEC TR 24766:2009 states that the following tasks within requirements engineering are processes that a requirements engineering tool should address, that is requirements elicitation, requirements analysis, requirements specification, requirements and product validation and verification, and requirements management [24].

2.4.1 Evaluation of Requirements Tools

Evaluating requirements engineering tools is greatly beneficial for companies when adopting a new requirements engineering tool [33]. A prevalent method of achieving this involves the quantitative comparison of the features of different tools based on a standardized criteria list. Such lists are typically derived from prior studies and are compiled by experts in the field. They encapsulate the comprehensive range of features that a requirements engineering tool should ideally possess, thereby providing a solid foundation for comparison. In this study, a combination of multiple criteria lists is utilized to examine the capabilities of T-Reqs, understand its strengths, and identify valuable features that may be lacking.

Santillán conducts an extensive literature review to examine various publicly available and industry-recognized criteria lists. Based on this review, they propose a new comprehensive list compiled from these sources. The sources of these lists are INCOSE, DSTO, ITEA, DESS, DaimlerChrysler, RWTH, Aachen, and Seilevel [30]. An overview of these lists is offered in Table 2.1. Seilevel, now ArgonDigital, has released a new evaluation report (2016) since the publication of the literature review was conducted with updated criteria [28]. Santillán's criteria list contains an extensive amount of features, some of which may not be applicable in all contexts; for example, features specific to software product line requirements were deemed

irrelevant within the scope of Ericsson's operations. Furthermore, while modeling was deemed unnecessary in their report, it has been reinstated as a relevant feature in this study. Additionally, due to the detail of the list, some overview is lost. For the purpose of this study, the list was therefore summarized into more high-level features. To avoid bias in the features selected in the summarized list, the AI model ChatGPT-4, developed by OpenAI [26], was used to maintain objectivity. The correctness of the items was then verified by the authors. Details pertaining to the prompt used for ChatGPT-4 can be found in Appendix C.

Publisher	Creation Year	Categories	Requirements
INCOSE	1990	13	68
DSTO	1997	11	149
ITEA DESS	2001	18	78
Daimler-Chrysler	2004	21	101
RWTH Aachen	2006	24	110
Seilevel	2011	23	230

Table 2.1: Criteria lists overview

Based on Santillán's compilation of popular criteria lists in industry and its summarization by an AI tool, the following feature list was devised (refer to Figure 2.2, 2.3, 2.4, 2.5 and 2.6) to facilitate the evaluation of T-Reqs against other prominent requirements engineering tools. The selection of these tools was based on the categorization of centralized, cloud-based, and decentralized tools. Rational DOORS was chosen as a representative centralized tool, while Jama represents the cloud-based category. T-Reqs and Doorstop exemplify decentralized tools. The information in regard to these tools' features was gathered from the websites, documentation, and Git repositories of the tools [4, 5, 46]. Additionally, ArgonDigital's 2016 requirements management tool evaluation report [45] was used, in which Jama was covered.

Although more features mean more possibilities, studies have shown that selecting a requirements engineering tool is not just a quantitative task, and it must conform to the context and environment of the operations in question [28,31,32,34]. The standard ISO/IEC TR 24766:2009 further adds to this sentiment by stating that; "Users tend to focus on specifying their functional or behavioral requirements, but users also have expectations about how well the product would work. Characteristics that fall into this category include how easy it is to use, how quickly it runs, how often it fails, and how it handles unexpected conditions. Such characteristics are known as software quality attributes or simply quality factors and are often considered as a part of the system's non-functional requirements." (p. 21) [24] A value-based evaluation approach is recommended by Heindl et al. [31], where the tool's features are weighted based on the value they bring to the company.

In this study, we heavily consider the context under the premise that T-Reqs is to be evaluated under real-world usage. In addition to the criteria list, we will discuss the features and factors that have been identified during qualitative interviews and

a follow-up survey with industry users. Based on users' statements, we will answer research questions RQ2: "What advantages and disadvantages of T-Reqs can be observed in practice in relation to traditional requirement tools?" and RQ3: "What characterizes tools such as T-Reqs in contrast with more traditional requirements tools?".

Internal requirements for the tool can also be defined to evaluate tools in accordance with the requirements engineering process and software tool environment at the company. Ericsson has developed a comprehensive list outlining the required functionality in terms of specific requirements. These requirements will be presented in Section 3.3, which will delve into the case context and prior research conducted at Ericsson. These requirements will serve as a basis for evaluating research question RQ1: "How well do the expectations of T-Reqs before its implementation match with how the tool performs today?", in addition to the responses in the interviews and survey.

Feature	**Description**	**T-Reqs (Decentralized)**	**Doorstop (Decentralized)**	**Jama (Cloud-based)**	**Rational DOORS (Centralized)**
(1/5) Requirements Management Main Activities and Baselining					
Version Control	The tool should provide features for reproducing and rolling back to earlier versions of requirements.	yes	yes	yes	yes (their own, i.e., not connected to the test cases and the code)
Change Control	The tool should be able to distinguish between formal and informal changes, provide a filter to review new/changed requirements, maintain an automatic audit trail for requirement changes, and notify project participants about changes via email. The tool should also make the history of requirements changes easily viewable.	partially (changes maintained with version control, view for history of requirements forthcoming)	partially (changes maintained with version control)	yes	yes
Status Tracking	The tool should allow requirements to be filtered by criteria (e.g., status), allow complex attributes with predefined value choices, and enable the editing of requirements' status.	yes	yes	yes	yes
Tracing	The tool should support linking requirements to existing documents, models, client documentation, notes, and emails. It should enable the creation of links between requirements of the same type, different types, and those in defined groups. It should be able to display traceability results in a table or a diagram, describe the nature of the relationship link between requirements, and enforce the creation and change of certain types of traceability links. The tool should support M:N bidirectional linking and the creation of a custom tracing model to link requirements to user-defined types of artifacts.	partially	partially (traces between items in yaml files)	partially	partially (usually links between artifacts or external links, not models unless including a modelling tool)
Baselining	The tool should allow for the creation of requirements baselines, comparison of baselines, and work with specific baselines. The tool should be able to show the difference, and what is new between the last version and the current version.	yes	yes	yes	yes
Modelling	The tool should enable the creation of requirement models.	yes	no	yes (limited linking due to being stored as images)	no

Table 2.2: Feature Comparison: Requirements Management Main Activities and Baselining

Feature	Description	T-Reqs (Decentralized)	Doorstop (Decentralized)	Jama (Cloud-based)	Rational DOORS (Centralized)
(2/5) Information Management					
Views	The tool should allow sorting and filtering of requirements based on various criteria, saving of custom views (both private and public), and instantly applying filtered views. It should support the display of requirements as a list, a hierarchy, or custom traceability data tables.	partially (filtering possible using python scripts, requirements displayed in a hierarchy)	partially (filtering possible using python scripts, requirements displayed in a hierarchy)	yes	yes
Navigation	The tool should offer keyword search, navigation through the requirements hierarchy, drag-and-drop ability for moving requirements within the hierarchy, and easy navigation of traceability hierarchy.	partially (no drag-and-drop ability)	partially (e.g., no drag-and-drop ability, search can be performed with python scripts)	yes	partially (unclear if it offers traceability hierarchy)
Analysis Functions	The tool should provide traceability analysis to identify missing links within requirements and detect inconsistencies in links.	yes	yes	yes	not clear
Import	The tool should support bulk entering of requirements, identifying requirements from external text documents, and batch import of structured data as new requirements from various types of documents.	yes	yes	yes	yes
Export	The tool should allow exporting requirements to various formats with the capability of highlighting changes against a previous baseline, exporting a subset of requirements based on a filter view, and exporting data from any report for further analysis.	yes	partially (e.g., no highlighting of changes)	yes	partially (unclear if it offers highlighting of changes)
Specification Generation	The tool should be able to produce requirements documentation in various pre-defined formats and allow defining custom formats for documentation output.	partially (no custom formats)	partially (no custom formats)	yes (difficult to get custom formats to look good)	partially (unclear whether custom formats are always offered)
Reporting	The tool should offer comprehensive reporting functionalities, including metrics reporting, metrics charting, reporting on requirement link problems, requirement maturity, review status, open issues, assignments, and more. It should allow viewing and customizing dashboard reports, creating burndown reports, generating traceability reports, and reporting on the number of reviews and issues of requirements.	forthcoming	yes, HTML	yes (burndown reports need integration)	yes (burndown reports need integration)
Users and Access Control	The tool should provide access management features to restrict access permissions for individual users and groups, and manage visibility of requirements for clients based on requirement attributes or project association.	no	not relevant (open-source tool providing CLI and connected to version control)	yes	yes
Offline Use	The tool should support working offline with the capability of merging changes upon reconnecting, providing reminders for synchronization, and offering a read-only view.	yes	yes	yes	no
User Interface	The tool should allow easy editing of requirements, support drag-and-drop linking of requirements, cut and paste functionality between the tool and other applications, and quick access to detailed information about requirements. The interface should be customizable, allowing assignment of functions to buttons, hot keys, and menus.	partially	partially (simple desktop and user interface)	partially (cannot drag-and-drop linking of requirements, cannot assign functions to buttons/hot keys/menus)	yes (considered dated however)
Usability	The tool should be easy to use and learn.	partially (developer friendly)	yes	yes	partially (extended documentation but buried functionality)

Table 2.3: Feature Comparison: Information Management

Feature	Description	T-Reqs	Doorstop	Jama	Rational DOORS
(3/5) Requirements' Data Management					
Requirements' Architecture	The tool should allow definition of data to be captured for each requirement, grouping requirements by project, organizing by groups and hierarchies, defining custom ID formats for requirements, managing dependencies and glossaries, tracking requesters, setting up complex attributes, capturing rationale, managing links to external documents, voting and setting priorities on requirements, and defining different types of requirements.	partially (e.g., setting up complex attributes)	partially (e.g., voting or prioritisation offered)	partially (no management for glossaries)	yes
Requirements' Capture	The tool should provide functions for easy entry of new requirements, assigning unique IDs, assignment of requirements to different analysts, marking new entries for review, copying requirements from other projects, defining stakeholders for each requirement, and allowing reuse of requirements between projects.	yes	yes	yes	yes
Requirements Edition and Deletion	The tool should allow editing and deletion of requirements with features for bulk editing, real-time updates, safe deletion options, and preventing concurrent editing by multiple users in online mode.	yes	yes	yes	yes
Requirements' Quality	The tool should offer automatic checks for spelling, grammar, and ambiguous words, and the ability to mark duplicate requirements and merge them together. It should also allow defining validation checks for user-defined attributes.	no	no	partially (no ability to mark duplicate requirements and merge them)	partially (unclear if identification of duplicate requirements is possible)
Requirements' Enrichment	The tool should support rich text formatting, tables, maintenance of formatting when pasting from other documents, inclusion of mathematical equations and special symbols, and incorporation of images and embedded documents. It should also support the addition of audio and video files.	partially (embedding in version control system)	partially (embedding in version control system)	partially (unclear if the addition of audio and video files is possible)	partially (unclear if the addition of audio and video files is possible)
Requirements' Issue Tracking	The tool should facilitate issue creation from requirements, link to issues in other tools, and have its own issue tracking functionality. It should also keep track of resolutions to open issues and maintain a record of open issues by requirement.	partially (issue tracking in review/version control system)	partially (issue tracking in review/version control system)	yes	unclear

Table 2.4: Feature Comparison: Requirements' Data Management

Feature	Description	T-Reqs	Doorstop	Jama	Rational DOORS
(4/5) Technical Specification, Licensing and Support					
Tool Integration	The tool should support automated integration with issue tracking systems, design tools, development environments, revision control systems for linked documents, note-taking software, and test management systems.	yes	yes	yes	yes
Extensibility	The tool should provide an external API and allow ad-hoc queries of requirement data.	yes	yes	yes	yes
System Specification	The tool should support both installed central repository and client-only installations. It should provide web and non-web interfaces, supporting Windows and Unix/Linux platforms, and manage multiple concurrent users. It should also detail platform resource requirements. The tool should support various database platforms including MS Access, SQL Server, Oracle, and MySQL, and allow the merging of requirements from different databases. It should also highlight its ease of installation and administration, and indicate whether it is open-source or commercial, and its current user base. It should scale with increasing number of input requirements and not limit the number of user-defined attributes.	partially/not relevant	yes	yes	yes
Licensing	The tool should provide information on its license policy and costs, warranty options, and maintenance & upgrade policies, along with maintenance costs.	not relevant (in-house development)	not relevant (open-source)	yes	yes
Training and Tool Help	The tool should offer a variety of learning aids such as sample requirements, workflow tutorials, training classes, and both online and offline documentation. It should also offer context-sensitive help within the system, phone support, and online support such as message boards, knowledge base, user groups, wikis, etc.	yes	yes	yes	yes

Table 2.5: Feature Comparison: Technical Specification, Licensing and Support

Feature	Description	T-Reqs	Doorstop	Jama	Rational DOORS
(5/5) Distributed Collaboration					
Informal communication	This includes the tool's ability to foster communication among stakeholders. This can be achieved through both synchronous (real-time chat, videoconferencing, VoIP) and asynchronous (email-like) means. The tool should allow users to insert links to relevant objects in their messages, access discussions they've been a part of, and save essential parts of a conversation's transcript.	no	no	yes (comment stream)	yes (in DOORS next generation)
Awareness	This pertains to the tool's ability to provide users with information about other team members. This includes their connection status, roles, geographical location, workload, and recent actions. The tool should also allow personalization of visual cues, send email notifications about actions taken by others, and allow users to subscribe to changes in requirements.	partially (provides awareness through version control system)	partially (provides awareness through version control system)	yes	unclear
Workflow management	The tool should be capable of defining custom workflows for different types of requirements, as well as for all types of requirements. It should also provide predefined workflows. The tool should track and request approval or sign-off for requirements, and enable task definition and rights assignment for each workflow step.	no	no	yes	yes (with integration to other IBM Rational tools)

Table 2.6: Feature Comparison: Distributed Collaboration

3
Case Context

In this chapter, we relate the previous research that has been conducted at Ericsson relating to their requirements engineering process along with more company-specific information, i.e., the context of this case study.

3.1 Previous Tools

The previous tools used at Ericsson were Rational RequisitePro and Rational Rhapsody. RequisitePro is a requirements management tool where requirements are written in Word documents and parsed into databases. Rhapsody is a modeling tool where unified modeling language, UML, is used and thus mainly suitable for functional requirements. Diagrams are created using preexisting shapes and symbols known to UML, which are dragged and dropped onto the canvas. Thus, the diagrams are highly customizable, and their layouts are configurable. An example image of Rational Rhapsody is provided in Figure 3.1.

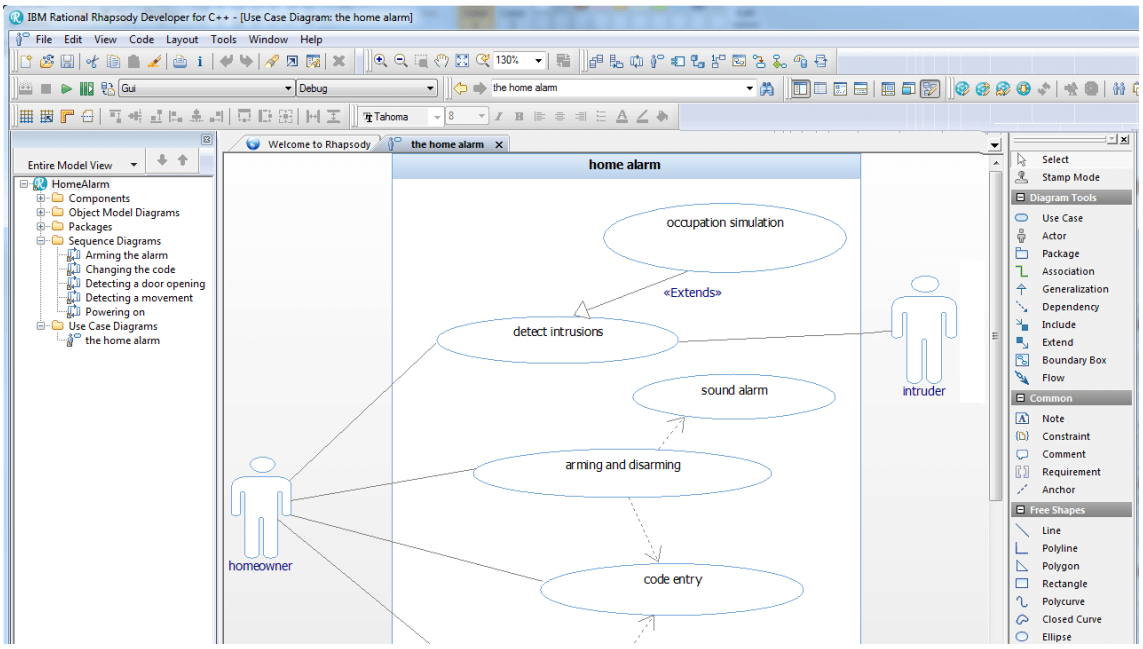

Figure 3.1: Example diagram in Rational Rhapsody from IBM's webpage; https://www.ibm.com/products/systems-design-rhapsody

In 2017 it was decided to replace these tools with a tool that better suited the intended practices at Ericsson. That is, the tools were not adequate for an agile en-

vironment, with cross-functional teams and continuous integration present. It must be noted that despite the tools not matching the desired ways of working, the main reason for the switch was that IBM was no longer supporting Rational RequisitePro, and the license cost of both Rational RequisitePro and Rhapsody was very high. In order to address the requirements engineering needs, a potential transition to IBM's alternative tool, Rational DOORS, was initially considered. However, an internal proposal to develop an in-house solution called T-Reqs was presented as an alternative. Following a comprehensive comparative study conducted within Ericsson, the T-Reqs proposal was ultimately chosen as the preferred option.

3.2 T-Reqs Tool Description

T-Reqs is a text-based requirements engineering tool, relying on Git-based knowledge storage and residing in a product's repository. Requirements are written and reside in Markdown files and are encompassed by XML tags which contain certain attributes relating to each requirement. Examples of attributes are the unique ID associated with each requirement and for which products this requirement has been implemented. A simplified example of a requirement is as follows:

```
<treqs-requirement id = "REQ100" implemented-for = "product">
Here is where the requirement is written
</treqs-requirement>
```

The Markdown files can also include trace links indicating traces between requirements and test cases, as well as PlantUML diagrams. These Markdown files containing requirements reside in the repository of each product and can thus be interacted with in a similar manner as normal code.

To increase the usability of the tool, different views and plugins have been implemented. The most notable are referred to as T-Reqs Web and T-Reqs Viewer and a plugin for the editors Eclipse and Visual Studio Code. The plugin dynamically displays the formatted result from the T-Reqs tags and PlantUML in the Markdown files as HTML.

T-Reqs Web (refer to Figure 3.2) is a web interface where requirements can be viewed. There, employees can search for requirements and have access to the ticketing system in JIRA, and they can see which feature a requirement belongs to and the implementation status of a requirement. They can also see the results of test cases and the requirements the tests are connected to.

T-Reqs Viewer (refer to Figure 3.3) is an interface that enables employees to make queries to an SQL database requesting a certain subset of the requirements that fit specific criteria. The database is only an intermediate storage for optimization, as during each build of the code, metadata contained in the product's repository will be uploaded to it.

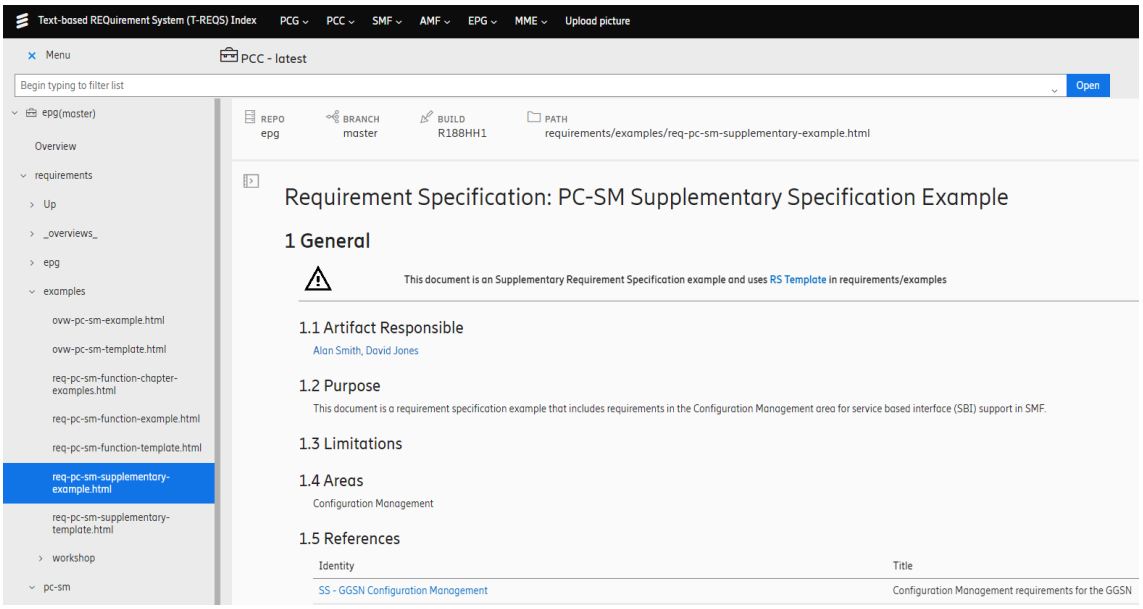

Figure 3.2: Example of T-Reqs Web interface

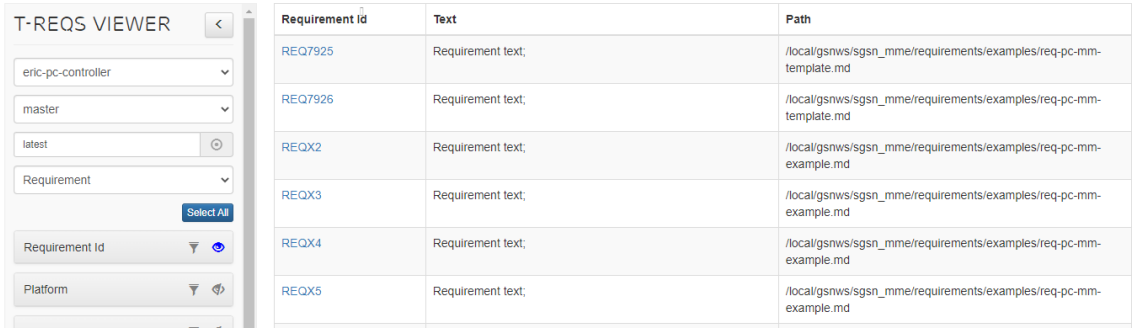

Figure 3.3: Example of T-Reqs Viewer interface

3.3 T-Reqs Related Research at Ericsson

A case study was performed at Ericsson to investigate the flow of requirements from strategy to release in large-scale agile, performed by Heikkilä et al. [10]. Data was collected in 43 interviews, which were analyzed qualitatively. The interviews were conducted in 2011 and 2013. The study illustrated the transition of the company from a plan-driven ways of working to agile development at a larger scale. The findings of the study showcased that practitioners noted increased motivation, improved efficiency, and increased flexibility. However, the noted problems were reported being overcommitment and growing technical debt, among others. The study mainly gives a glimpse of how these procedures worked before the usage of the current requirements engineering tool and thus offers a comparison to the current state of affairs.

As mentioned above, different challenges have been identified when performing requirements engineering in agile software development. According to Knauss et al. [7], the challenges most relevant to T-Reqs are as follows,

C1 Updating and deprecating requirements
Requirements defined at the beginning of development can become outdated and thus need to be changed. The challenge arises if it is uncertain who should update the requirements and when those updates should take place.

C2 Access to tooling and requirements
Traditional requirements tools often charge fees per user. Thus, it does not facilitate multi-user access for companies performing large-scale agile processes.

C3 Consistent requirements quality
It is a challenge to keep the quality of requirements consistent across different artifacts.

C4 Managing experimental requirements
It is important to capture experimental requirements as they might later be incorporated into the design. The challenge is that these requirements must be handled differently than stable requirements.

C5 Create and maintain traces
Traces are important in order to determine whether a requirement has been implemented or not. However, they are not always prioritized by the development team as they provide no direct value.

C6 Plan verification and validation based on requirements
The challenge relates to how testing can be planned when requirements are now incomplete and incremental.

In the pursuit of selecting a tool to mitigate these challenges, Ericsson had specified requirements, two of which could not be fulfilled by most requirements engineering tools on the market in 2017. They read; (1) The tool must integrate well with Git, (2) The tool must allow for changes of the same artifact by many people [7].

Other requirements relating to the tool included that it must be feasible to store the traces from requirements to test cases in order to know when requirements were delivered as code and how it was tested, as well as to know which requirement was delivered at a certain time. The traces should be stored in the exact location of the other artifacts to assert that the trace information is always in sync with the artifacts. In terms of the requirements themselves, it must be possible to assign a unique identifier to them as well as store additional descriptive information about them, i.e., attributes. It should be a possibility to view the requirements in a human-readable format. Additionally, both requirements and test cases, including their definitions, should be stored in the same Git repository as the code. It must also be possible to create and store test results. Further, changes to the artifacts should be easy to undo as well as easily updated by the cross-functional development team. The selected tool should also be similar to the current tools being used by the team. Finally, it needs to be possible to validate the system after making changes, that is, make certain that no edit can break the system and it should also be possible to release any build with minimal manual work, as requirements and test cases should always be in sync with the code.

Since the traditional tools could not support the desired functionality in regard to large-scale agile practices, Ericsson eventually decided to develop a tool

in-house to account for their situation and environment. Thus, T-Reqs was developed, and with textual modeling of requirements engineering, it was expected to mitigate some of the challenges met within requirements engineering in large-scale agile environments.

3.4 Roles

The roles within the company that are relevant for this case study due to their proximity and usage of T-Reqs are General System Manager and Team System Manager. Thus, an explanation of the responsibilities of said roles will be given to emphasize the difference between them and give better context behind the perspectives of the interviewees later in the report.

General System Manager: The responsibility of the General System Manager is twofold. Firstly, to perform studies, i.e., look ahead and discover which feature or functionality should be implemented and how. These features can be based on the mobile broadband standard, which Ericsson follows, or customer requests. These studies are either classified as system studies or initial feature studies. The system studies have a broader scope and are often later broken down into multiple initial feature studies. The scope of the initial feature studies corresponds to a feature that a cross-functional team would be able to work on. Upon finalizing the decision to proceed with the development of a specific feature, the preparation of system requirements for the feature development in the teams falls under the responsibility of the General System Manager as well. This feature is then presented to the team, and the General System Manager supports the team with creating and writing the requirements from the initial, often more high-level requirements. Thus, breaking down the requirement into more technical requirements. The General System Manager continuously reviews the requirements throughout the development process of the feature and is notified of any changes made by the team. The collaboration with the team is mainly done through the Team System Managers, and each feature a team works on has an assigned General System Manager. Secondly, the General System Managers have the responsibility to answer questions from customers and business salespersons, in addition to handling change requests from customers.

Team System Manager: The Team System Manager is part of the cross-functional development team and thus is either a developer or functional tester alongside their role of Team System Manager. Their responsibility is to understand the feature, write and update the requirements in T-Reqs, support the team by explaining attributes or fields to other members of the team. Additionally, they have discussions regarding the requirements with the team and the General System Manager responsible for the feature as well as communicating any changes made during development to the General System Manager. The Team System Manager thus has the perspective of the requirements in mind, but the whole team is responsible for the requirements. It is important to note that despite the General System Manager being the artifact responsible for the requirements, the team must be able to defend the requirements and ensure that they correspond to what has been implemented.

4
Methods

The method chosen for this master's thesis is an in-depth case study [10]. The design chosen for data collection is inspired by Creswells' proposed mixed method research and design [11]. In this design, quantitative and qualitative data and methods associated with said data are combined. Among the different types of mixed method designs suggested in Creswells' book, the chosen method for this research is Exploratory Sequential Mixed Methods Design which occurs in three phases. The initial phase entails exploring qualitative data. The second phase entails a feature being created, in our case, a survey instrument, from the gathered qualitative data. Finally, in the last phase, the feature is tested by using quantitative research methods.

This case study method was chosen to get a good understanding of the context and the people who interact with T-Reqs, in order to give a more accurate evaluation of T-Reqs. The decision to use an Exploratory Sequential Mixed Methods Design, starting with interviews and then a survey, is based on the idea of having a good overview of the themes present in the context before quantifying them with a survey, which is also standard practice [35]. The objective is to create a comprehensive survey that thoroughly encompasses all facets pertaining to the research inquiries while remaining vigilant in avoiding any potential influence from prior biases in the construction of the survey framework. The outcomes obtained from the survey can serve as a valuable instrument to assess and evaluate the themes identified during the interviews and, subsequently, establish a collective agreement among the interviewees. The overview of the case study process can be seen in Figure 4.1.

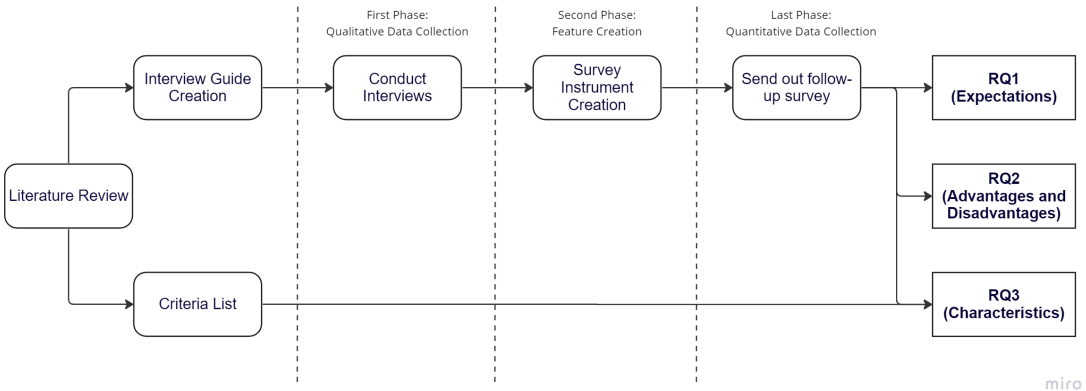

Figure 4.1: Overview of the case study

4.1 Data Collection

As per the chosen design, data will be collected in the first phase and the last phase. In the first phase, data was collected through semi-structured interviews, i.e., a data collection method associated with qualitative research. In the last phase, a collection of data in a follow-up survey, i.e., a method connected to quantitative research.

In addition to the data collected through the chosen design, the preliminary results from the interviews were presented at a workshop with requirements engineering practitioners. This was in order to achieve more generalization of our findings in this case study by getting insights from practitioners whether or not T-Reqs would be applicable in their company context and what advantages and characteristics of T-Reqs resonated with them.

4.1.1 Interview Methodology

The research design incorporated a series of fourteen interviews as a primary data collection method. Participants were selected based on their roles within Ericsson, comprising both General System Managers and Team System Managers with different levels of experience, reflecting a range of perspectives on the use of the T-Reqs tool. That is, all participants are the current users of T-Reqs and are thus familiar with the tool and have experience with working with it.

The selection of interviewees followed a mixed approach that incorporated elements of convenience and snowball sampling strategies [36]. Initially, our company liaison provided a list of potential participants amenable to participating in the study. Subsequently, we leveraged the snowball sampling method, asking these initial participants to recommend additional personnel within Ericsson who could provide valuable insights for the research. This dual strategy ensured a broad and diverse range of views, enriching our understanding of T-Reqs within the company context.

Interviews were conducted until perceived saturation was reached, that is until no new themes emerged. At the same time, the sampling of the interviewees with snowball sampling had reached its end. That is, we had no immediate contacts as possible next interviewees.

The interviews were semi-structured, which entails that an interview guide was prepared for the interviews with specific questions to ask but allowing for improvisation as well. It is hard to know beforehand all relevant topics or areas that might be valuable to discuss. Thus, it is beneficial to leave room for other topics of discussion relating to an answer to the research questions.

The interviews took approximately one hour and were structured by beginning with identifying the background of the interviewee, i.e., their current and previous roles, how long they had been employed at Ericsson, and if they had any experience with other requirements engineering tools. Then, the questions posed related to each of the research questions in succession. Beginning with collecting data on how well the tool is performing in terms of meeting the desired expectations and whether it is solving the relevant problems they had with the previous way of writing and managing the requirements. Additionally, discovering if there are any unexpected

Subject	Current Role	Experience
Interviewee 1	General SM	High
Interviewee 2	General SM	High
Interviewee 3	Developer	Low
Interviewee 4	General SM	High
Interviewee 5	General SM	Low
Interviewee 6	General SM	Low
Interviewee 7	General SM	High
Interviewee 8	General SM	High
Interviewee 9	General SM	High
Interviewee 10	General SM	Low
Interviewee 11	General SM	High
Interviewee 12	General SM	Low
Interviewee 13	Team SM	Low
Interviewee 14	Team SM	Low

Table 4.1: Table of interviewees

benefits or problems with the tool. Next, which advantages and disadvantages the interviewees perceived the tool to possess. The participants were finally asked to identify either the characteristics of the tool or if the interviewee was not familiar with other requirements engineering tools, the key features of the tool. The interview guide devised and used for the interviews can be seen in Appendix A.

In Table 4.1, the interviewees are listed with both their current role and their experience level. The column indicating experience is associated with their experience with using other requirements engineering tools. High experience indicates that the interviewee had experience with working with other requirements engineering tools. This correlated with the interviewee having over five years of working experience at Ericsson, i.e., before switching to T-Reqs, and thus they had experience with Rational RequisitePro and Rhapsody. Low experience indicates that the interviewee's only experience with requirements engineering tools is through using T-Reqs. This correlated with the interviewees having joined the company after the switch to T-Reqs.

4.1.2 Survey Methodology

The qualitative results from the interviews were used as input for constructing the survey questions. This increased the possibility of more precision and less ambiguity due to the questions being crafted with specific domain knowledge and vocabulary used by the interviewees. The ambiguity is further decreased due to the survey being sent out to the participants of the interviews, i.e., as a follow-up survey. Thus, the respondents of the survey were only employees that had participated in the interviews. Ten out of fourteen possible respondents answered the survey. The rest were not able to answer due to unavailability, and one participant did not answer due to the nature of their interaction with the tool.

The reason for gathering quantitative data from the same employees as had

Subject	Current Role	Experience
Respondent 1	General SM	High
Respondent 2	General SM	Low
Respondent 3	Team SM	Low
Respondent 4	Team SM	Low
Respondent 5	General SM	Low
Respondent 6	General SM	High
Respondent 7	General SM	High
Respondent 8	General SM	High
Respondent 9	General SM	High
Respondent 10	General SM	High

Table 4.2: Table of respondents to the follow-up survey

been a part of the qualitative data-gathering phase was twofold. First, by sending the survey only to the previous interviewees, the likelihood of them misunderstanding the questions was reduced as these were the topics discussed in the interviews. Second, this made it possible to verify that a topic or opinion brought up by only one interviewee was shared among the other interviewees. That is, to get an understanding of other interviewees' sentiments towards the topics or opinions, whether they found it irrelevant or had not thought of it during the interview.

The structure of the survey resembled the structure of the interviews, as participants were first asked to indicate their current role at Ericsson, as well as whether they had previous experience with other requirements engineering tools. Additionally, they were asked to specify their general IT competencies, i.e., how comfortable they were with using Git, Linux, and PlantUML. Table 4.2 shows the respondents to the survey. The IT competence of each respondent has not been added to the table as each participant either agreed or strongly agreed with being comfortable with all three technologies, except *Respondent 3*, which disagreed with being comfortable with using PlantUML. After the questions relating to the background of the participants, questions were posed where the participants were asked to relay their sentiments to different statements regarding the tools. The statements were grouped based on relevant themes and topics identified during the interviews and were ordered based on which research questions they answered. Most questions in the survey were in the format of a 5-point Likert scale, ranging from strongly disagree to strongly agree. The option of "I don't know" was also afforded to the participants. Apart from the Likert scale questions, multiple answer questions were also posed in order to validate the identified advantages and disadvantages as well as the characteristics and key features of T-Reqs. The survey was online and sent out to the participants of the interviews via email to minimize the disturbance in the environment. The survey questions can be viewed in Appendix B.

4.1.3 Workshop Methodology

Our preliminary findings derived from the interviews were presented to experts in the fields of requirements and requirements tooling from different companies in Gothen-

burg at a workshop at the Software Center at Chalmers University of Technology. Two participants were from Ericsson, two from different automotive companies, and one from a manufacturing company, along with two academic researchers from the Computer Science and Engineering department at the University of Gothenburg and Chalmers University of Technology. All participants had some prior knowledge of T-Reqs.

The findings were presented in a slideshow presentation and followed by an open discussion among the participants of the workshop. The purpose of this additional data collection was to achieve more generalization of our findings. That is, which identified advantages, disadvantages, and characteristics of T-Reqs the workshop participants resonated with based on their company context and whether the tool would be suitable for their development processes. Thus, gaining a broader perspective of which conditions and contexts T-Reqs would be most applicable.

4.2 Data Analysis

The analysis of the semi-structured interviews was conducted using coding methods. In this context, coding refers to the process of categorizing and tagging raw data from interviews or surveys to identify themes, patterns, and relationships [13]. Coding was performed after the interviews, i.e., no code was predetermined. Additionally, coding was performed independently by each author to ensure that no information relayed in the interviews was overlooked and potentially identifying different perspectives of how the data could be interpreted. The codes from each of the authors were compared, and in most cases, they relayed the same information, however, with different degrees of detail.

We used a combination of coding methods suggested by Saldaña in his coding manual [13], specifically First Cycle methods. These methods are foundational coding procedures that are typically applied during the initial stages of qualitative data analysis. Saldaña suggests using In Vivo Coding, Initial Coding, and/or Values Coding specifically for interview transcripts. We found Initial Coding and In Vivo Coding to be most suitable for our data. This was mainly due to Initial Coding offering the freedom, in the beginning, to code as we went and try to give the best representation of the data, i.e., as Saldaña states, Initial Coding is the first phase of approaching the data through grounded theory. In Vivo Coding further offers to give the best representation of the data as it involves using a word or short phrase from the actual language of the participant as a code. It is useful for capturing how participants frame their experiences using their own terms. For the final suggestion, i.e., Values Coding, we did not find it as intuitive as the other two coding methods. Values Coding involves associating a code either with the letter V, A, or B, indicating whether the code relates to the interviewee's value, attribute, or belief. As was mentioned in Saldaña's manual, it can be hard to determine to which type an interviewee's remark should be classified. We also thought that we risked that our own beliefs and thoughts would affect the way that we categorized the remarks of the interviewees. Therefore, we would not be offering the best representation of the data.

As we progressed with the coding, we noticed elements of Process Coding

emerging in our codes. Process Coding involves using gerunds ("-ing" words) to identify observable and conceptual actions in the data. This correlated with the coding manual, stating that Initial Coding could often include In Vivo and Process Coding.

Given the diverse range of topics covered in the interviews, adding another layer of specificity to our codes was often necessary to clearly indicate, for example, which tool the comment related to. To achieve this, we employed Subcoding. This coding method involves assigning additional codes to a parent code to provide extra details or capture a specific aspect of the coded data.

In summary, we used a combination of Initial Coding, In Vivo Coding, Process Coding, and Subcoding methods to thoroughly analyze the data collected from the semi-structured interviews. These coding techniques allowed us to distill the rich qualitative data into identifiable themes and patterns, ultimately enabling us to draw meaningful conclusions from our research.

The following are examples of codes from the interview transcripts:

> "One goal was, [...], the usability. That it should be fairly easy to set up and to use the tool on a daily basis" - Interviewee 1

The code associated with the quote above is: *T-Reqs goal – Usability, "easy to set up and to use"*. In the code *T-Reqs goal* is the Initial Coding, *Usability* is the Subcoding, and *"easy to set up and to use"* is the In Vivo Coding.

> "Initially, it was very tough because if you want to copy the whole image, it was not quite easy. You had to choose and select and so many things was, yeah. Sometimes it gets wrong and you know copy pasting the things doesn't work, it didn't work properly. So they had kind of a lot of issues, yeah." - Interviewee 4

The code associated with the quote above is: *Rhapsody: Issues with copying image*. In the code *Rhapsody* is the Initial Coding, and *Issues with copying image* is Process Coding.

The evaluation criteria of the tool will be based on whether the tool has met the desired expectations and solved the problems it was intended to solve. This can be retrieved by comparing the answers in the interviews and the survey to the documentation stating these expectations and mitigation of problems that justified the initial decision to choose to work with T-Reqs.

4.3 Ethical Considerations

The following measures were taken to follow ethical guidelines regarding human participants. The purpose of the interviews and the document containing the original proposal for this research were provided to the participants prior to the interviews. The purpose of the survey was also relayed. Participation in the interviews and survey was both voluntary. The names of the participants were not recorded or disclosed in any of the research documents. Thus, anonymity and confidentiality were guaranteed. The interviews took no longer than one hour, to cause the least

amount of disturbance to the participants' daily work. The participants were also offered to ask questions if they needed any clarifications. Verbal consent for recording the interviews was given at the start of each interview.

4.4 Threats to Validity

Given the research method chosen and the data collection and analysis performed in this research, certain threats to validity have been introduced. In this section, these threats will be discussed in accordance with the classification suggested by Runeson et al. [11], i.e., construct validity, external validity, internal validity, and reliability.

4.4.1 Construct validity

A threat to construct validity that is present in this research concerns the topic of the tool in comparison to the previous requirements engineering tools at Ericsson. That is, half of the interviewees were not familiar with the previous tools or other tools in the industry except T-Reqs. This posed a threat since two research questions rely on knowledge of traditional requirements engineering tools. For the second research question, where we asked the interviewees to describe the advantages and disadvantages of the tool, this could be done without the knowledge of other tools. However, for the research question regarding the characteristic of the tool, i.e., its unique features, it proved more challenging. Thus, the interviewees that did not have knowledge of other tools were instead asked to identify what they considered the key features of the tool.

In addressing potential threats to the construct validity in our research methodology, it is crucial to acknowledge the role of the generative AI, ChatGPT-4, used in our study. A key area where the AI tool was utilized was in summarizing a comprehensive criteria list from a prior study [30]. While summarizing, we were conscious of the potential for bias towards T-Reqs if the summarization was performed manually by the authors, as we might select items in the list in favor of T-Reqs, due to our closeness to the tool. To mitigate this potential bias, we deployed the AI tool. However, using an AI introduces its own potential threats to validity. Specifically, there might be subtle changes in the meaning of the text due to the AI's interpretation, posing a risk to the construct validity. To counteract this, we have diligently reviewed the AI-generated summary and made necessary adjustments to ensure the correctness and integrity of the summarization.

The AI tool was also employed for grammar and syntax correction in parts of the paper. While the tool generally improves the consistency and clarity of the text, it may not perfectly understand the context of the study. It could suggest changes leading to inaccuracies in the grammar or phrasing. Therefore, although we carefully scrutinized all AI-suggested changes, there is still a possibility of minor discrepancies that might affect construct validity. These potential issues are essential to consider, though they are unlikely to significantly impact the overall findings or conclusions of our study.

4.4.2 Internal Validity

Given the heavy reliance on semi-structured interviews in this study, there is a risk that the authors' expectations and preconceptions could influence the data analysis process. To address this potential threat to the study's validity, we adopted a coding approach based on established strategies proposed by Saldaña [13]. The interview transcripts were coded independently by both authors and then compared to promote consistency and reduce subjectivity. However, another potential threat to the reliability of our study comes from the absence of checks for the coding process by an independent, external party. This lack of external verification could introduce a risk of the codes being less accurate or reliable, as the authors are not experts in the field and have limited prior knowledge of requirements engineering tools. Although this risk was partially mitigated by the independent coding process performed by the authors, the absence of an external reviewer remains a limitation. To verify and validate our findings from the interviews, we sent out a follow-up survey to the interviewees in order to confirm our interpretations of the data gathered from the interviews. Through these measures, we sought to enhance the rigor and reliability of the study's qualitative data analysis.

By using convenience sampling, a potential bias can be introduced, i.e, sampling bias, as the ones agreeing to be interviewed might have a more positive view of T-Reqs than the average population. Consequently, the sample may not be representative of the entire population and limits the generalizability of the study's findings. To address this limitation, we employed several strategies. First, the sample size was selected with the goal in mind of striving to achieve data saturation, i.e., get as many perspectives as possible in order to draw necessary conclusions. Second, we utilized snowball sampling to diversify the participants and obtain a broader range of perspectives, including hard-to-reach subjects. Third, to obtain a more diverse sample, we also compared the characteristics of the participants to those of the population. Specifically, we focused on the participants' role at Ericsson and their experience with requirements engineering tools as the main criteria for inclusion in the study. By considering these characteristics, we aimed to ensure that the sample accurately represented the target population and increased the generalizability of the study's findings. Despite these efforts, sampling bias remains a potential threat to the validity of our findings.

4.4.3 External Validity

The delimitation of this study includes reducing the scope of the case study to focus mainly on the text-based requirements engineering tool at Ericsson instead of performing a general case study on said tools. This decision was mainly due to resources and time constraints, i.e., depth was preferred over breadth, considering the qualitative nature of a more complex evaluation like this. The authors of this study also had the opportunity to work with T-Reqs during the master's level course Industrial Practice Project in Software Engineering and familiarize themselves with the tool.

Due to the delimitation of the scope of the case study, the main limitation of this study is the lack of generalizability. Since it is an evaluation of a tool in a

contained environment, i.e., within one company, the possibility could arise that the company's experience with the tool could not be comparable to another company.

The inherent risk of limited generalizability posed by a single case study approach was mitigated through several strategic measures. Primarily, this was achieved by formulating a research question that expands the scope to encompass the general characteristics of these new tools and proposing a definition for them. Additionally, a workshop was conducted to engage a diverse group of practitioners in the discussion of preliminary findings derived from the interviews. The goal of these discussions was not only to validate and refine the insights gained from the Ericsson context but also to extrapolate these insights to broader contexts. By facilitating a dialogue regarding the potential adoption of T-Reqs in other companies, we strived to identify conditions and contexts where T-Reqs could demonstrate utility. This method provided a platform for participants to share their unique perspectives, thus enriching the research findings and enhancing their potential applicability beyond the specific case of Ericsson. Finally, from the findings of this study, we offer advice for what should be considered when choosing a requirements engineering tool in Section 6.2.

By illustrating the specific context in which T-Reqs has been employed, considering the fulfilled expectations, advantages, and disadvantages of the tool, we assume transferability [42] can be established. That is, experts in the field of requirements and tooling can interpret the results of this study in order to evaluate if they are applicable in their contexts. Thus, the workshop could be used to assess whether transferability had been achieved.

4.4.4 Reliability

There exists a threat to the reliability of the research due to it potentially being hard to replicate as the interviews were semi-structured. In order to minimize this threat, the results derived from the interviews were verified by sending a follow-up survey to the interview participants.

A threat to reliability is also present in terms of the questions posed during the interviews or in the survey being unclear. This ambiguity could result in a misunderstanding which will evidently skew the results. To mitigate this threat, the authors performed a literature review analyzing challenges found regarding requirements engineering tools to familiarize themselves with the terminology and features most common to the tools. This vocabulary was then employed to form the questions for the interviews. Additionally, due to the authors being acquainted with the tool from previous interactions with it, the terminologies and vocabulary employed in relation to it at Ericsson were already familiar to the authors and enabled easier discussions in the interviews, minimizing misunderstandings. For the questions posed in the survey, this threat was mitigated by asking questions based on topics and themes discussed during the interviews. Additionally, the survey was only sent to those who participated in the interviews.

5
Results

In this chapter, the results of this case study will be presented in accordance with the previously posed research questions.

5.1 RQ1 (Expectations)

How well do the expectations of T-Reqs before its implementation match with how the tool performs today?

> The main goals and expectations for the tool were increased usability and improved ways of working. Most interviewees that were aware of the original goals of the tool agreed that the tool had achieved what it set out to achieve for the most part. Several interviewees noted that they had seen an increase in requirements updates done by the teams themselves, the scale-out of the company was successful, and the tool is currently being adopted in organizations across the company.

To answer this research question, we start by describing the considerations and decisions made regarding the implementation of T-Reqs, gathered from both the interviews and artifacts provided by Ericsson. Then relate the different expectations and goals concerning the tool and whether those are considered to have been met or not.

5.1.1 Implementation Considerations & Decisions

The following main areas of consideration were identified when it came to implementing T-Reqs, i.e., version control and storage, requirements modeling, traceability of requirements and test cases, interoperability, ways of working, and other tools.

5.1.1.1 Version Control & Storage

When using the previous requirements engineering tools, i.e., Rational RequisitePro and Rhapsody, the version control handling had to be done manually and separately from the tools. This evidently introduced more manual labor and was not time efficient. It was thus considered important to be able to store requirements, code, and test cases together to enable an easier way to baseline and thus compare and review changes to both requirements and the code and test cases simultaneously. In order to store requirements together with the code, it was considered most convenient to

store the requirements in text-based files, i.e., Markdown files, which also complied with the need for being able to view the requirements in a human-readable format.

5.1.1.2 Requirements Modeling

In order to replace the previous modeling tool Rhapsody, T-Reqs had to have the possibility of also displaying diagrams alongside the requirements. It was decided to integrate PlantUML into T-Reqs as it is text-based as well and could thus be included in the T-Reqs files and managed the same way as the requirements.

5.1.1.3 Traceability of Requirements & Test Cases

The storage of trace links between requirements and test cases is vital as they justify whether a requirement has been fulfilled or not. Thus, illustrating when a requirement was delivered as code and how it was tested. The original idea was to still pertain the tracing information and the metadata in a database, as was done in Rational RequisitePro. However, it was instead proposed to include the trace information in the requirements documents, and test case files themselves. Therefore, all artifacts are in a similar location, and in order to add a new requirement, the update could be made in only a single file. This will also assert that the trace information is always in sync with the artifacts.

5.1.1.4 Interoperability of T-Reqs

Different organizations within Ericsson have varying products that they develop, as with varying tools that they employ. Therefore, it was important that the tool had interoperability with other systems in use. The byproduct of selecting a traditional requirements engineering tool is that they are often part of a package of other tools, which the companies creating these tools suggest are used collectively. However, Ericsson already had these other systems for their development processes, e.g., test, build, and issue reporting systems, which they did not want to abandon without cause. Thus, it was important that the new tool fit the current tool suite.

As mentioned above, it was important to have version control as part of the tool, and thus, in order to make versioning consistent, T-Reqs is integrated into each product's repository. This was not done all at once and has rather been a gradual spread across the departments and products, and T-Reqs plugins implemented for the different development tools each department relies on.

5.1.1.5 Ways of Working

Apart from needing to switch tools due to RequisitePro no longer being supported, one of the major factors in changing tools was also due to a new way of working, which Ericsson wanted to introduce. Namely, they were moving towards more agile ways of working. In doing so, they wanted to involve the team more in the requirement engineering process, i.e., integrating it more into the development process and increasing the teams' responsibility of managing the requirements. This should also enhance the teams' understanding of the requirements and aid in the implementation. This would be a challenge when considering licensing tools, as they are often

charged per user. Thus, an incentive exists to either pick an open-source tool or develop a tool in-house.

5.1.1.6 Other Tools

At the time, Ericsson experts evaluated other tools on the market, but none were considered suitable for the development at the company, which opted thus to develop its own tool. As many interviewees pointed out, this is not an uncommon occurrence as there is a culture within Ericsson to develop their own tools. Which, as one interviewee stated, has been done with varying success, but they considered T-Reqs to be a *"decent result" (Interviewee 6)*. On the other hand, another interviewee stated that they didn't know if in-house development was good or bad as a lot of the initial work performed on developing the tool *"could have been resolved with having a licensed tool" (Interviewee 2)*. However, the consensus among the interviewees was that developing the tool in-house was a good decision as it was then tailored to Ericsson's needs, i.e., licensing tools offer a large variety of features that are not all applicable for Ericsson's requirements management. Additionally, it makes it easy to reach out and request changes or bug fixes, i.e., offers *"good customer support" (Interviewee 10)*. However, *Interviewee 14* noted that compared to the licensing tool, T-Reqs lacks in providing extensive documentation and instructions, though that is on the mend.

5.1.2 Goals & Expectations

The following goals and expectations towards T-Reqs were identified, i.e., usability and improved ways of working. The expectation of improved ways of working contains the subgoals scale-out, automated test reporting, facilitated collaboration, pertain the structure, and master branch release ready.

5.1.2.1 Usability

One of the primary goals of introducing the new tool was to emphasize ease of use, i.e., that it should be easy to set up the tool as well as to use it. This should be true for both frequent users and infrequent users, i.e., the tool should also be intuitive enough to benefit users that do not use it on a regular basis. This was partly solved by using the existing Git framework accessible through a Linux virtual machine, i.e., technology that most developers were familiar with. This familiarity with Git also lends itself to the expectation of developers having less fear of losing work, i,e., for example, changes made should be easy to undo. Using the existing Git framework also ensured that developers could remain working in the same environment, i.e., that they did not have to switch between tools while developing. In addition to the version control, an HTML preview can be generated in the editor to parse the syntax written for the requirements in the Markdown files, including the PlantUML diagrams. This enables the developers to view how the requirement will be displayed in the T-Reqs Web before they commit their changes.

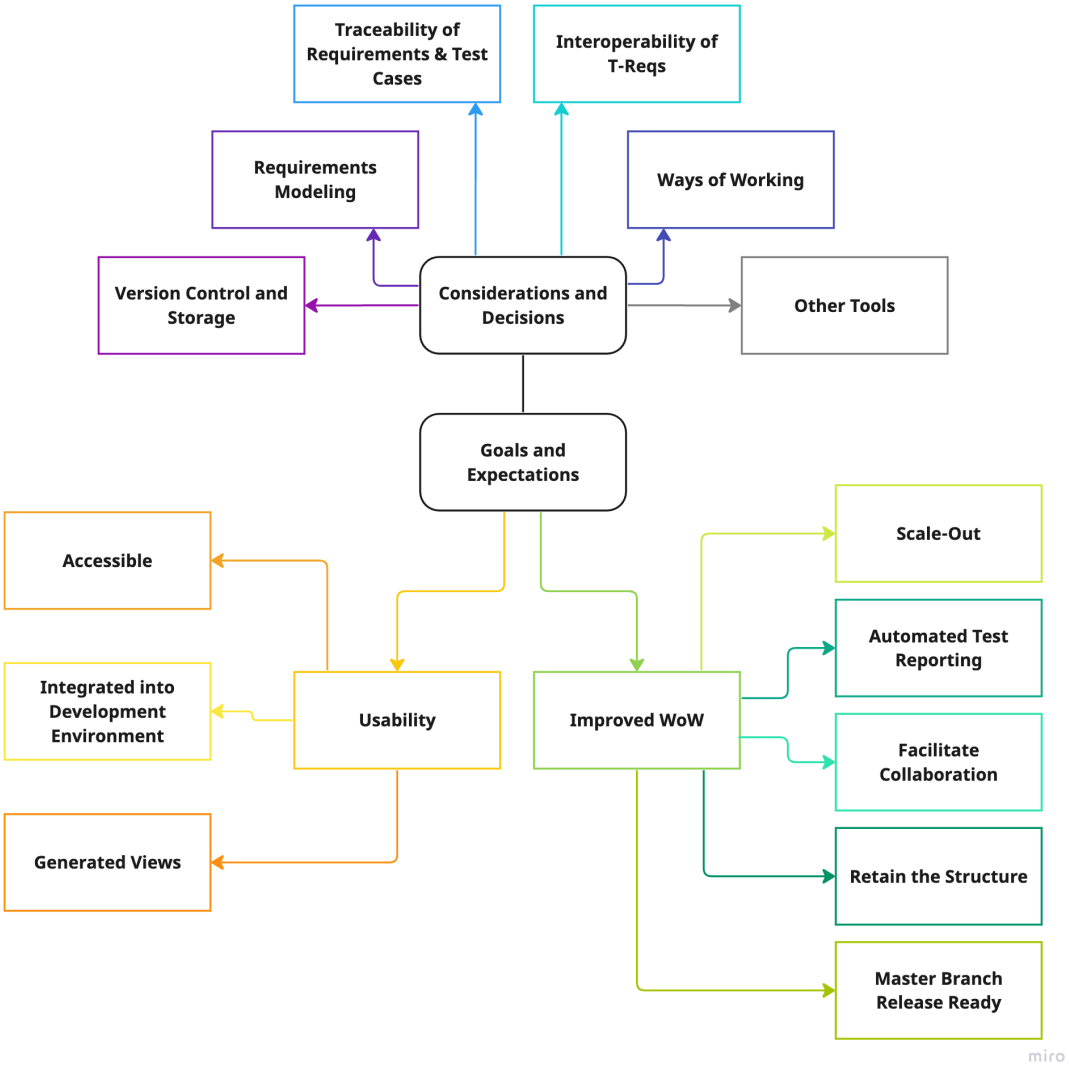

Figure 5.1: Visualisation of the themes and sub-themes identified in regard to RQ1 (Expectations)

5.1.2.2 Improved Ways of Working

Scale-Out: One of the main expectations of moving the requirements closer to the development teams was to be able to scale better, i.e., increase the number of cross-functional development teams, thus enabling large-scale agile development. For Ericsson's previous ways of working, it was the responsibility of the General System Manager to make sure that the requirements were written and up-to-date. Due to there only being a limited amount of General System Managers, then by introducing new development teams, Ericsson risked creating a bottleneck in their requirements management processes. That is, the teams would have to wait longer for feedback from the General System Managers resulting in delayed development or deprioritization of requirements. However, by moving the requirements closer to the team, this bottleneck is eliminated, or more specifically reduced, as the team still needs to notify the General System Manager of any changes as they are the artifacts responsible.

Moving the requirements closer to the cross-functional development teams also enables the contribution of everyone in the team. Thus, instead of having *"big bangs" (Interviewee 8)* there are more frequent and smaller changes. Which, in turn, makes it both easier to maintain the requirements as well as update them. This also makes it less likely that the requirements are ignored, i.e., if the team developing the code is the one responsible for keeping the requirements and code in sync.

Automated Test Reporting: Another primary goal was to bring down the cost of manual test reporting. Previously, that is in Rational RequisitePro, the test results had to be manually written in a database after the tests were run. It was possible to automate this process when the traces from test cases to the requirements were introduced in the test case files, as mentioned above.

Facilitate Collaboration: One of the challenges that present itself in large-scale agile development relates to collaboration, more specifically, how to bridge the gap between plan-driven and agile development (Section 2.2.1). In the case at Ericsson, this corresponds to the collaboration between the General System Manager and the development team, particularly the Team System Manager.

The consensus among the interviewees was that this collaboration was going well, as remarked by a General System Manager,

> "...I feel it's quite easy to just exchange information with [Team System Managers] and request changes or come to an agreement of who should make certain changes for documents and so on." - Interviewee 6

The collaboration was also considered to be facilitated with the help of Git and Gerrit, i.e., the reviews in Gerrit offer transparency as changes in the documents are displayed in an easily readable manner. The review process with the previous tools was a lot more challenging. In Rhapsody, this was due to the fact that it was hard to see what had been changed and when, while in RequisitePro, the documents were sent via email for review. Thus, as one interviewee stated,

> "So out of the tools we have used, I would say this is the best one for collaboration" - Interviewee 9

T-Reqs Web was also noted by one of the interviewees as a good point of discussion between the system managers, i.e., for showcasing the written requirements. However, one interviewee pointed out that they considered that the facilitation of the collaboration was less due to the T-Reqs tool and more due to the ways of working. In addition, if the rules established by the way of working were being followed, i.e., that the right people were being notified of certain changes.

The collaboration between the Team System Manager and the development team they are a part of seemed to be more challenging in some cases when it came to requirements management. This was illustrated by one of the Team System Managers, by stating that they felt as if they were the only ones in the team that cared about the requirements. Another interviewee stated that there was a lack of engagement from the other team members but that the team did a final review together. If the need for a change in the requirements was noted by a team member, the protocol is to raise the issue with the Team System Manager, that makes the changes in T-Reqs and notifies the appropriate General System Manager.

Retain the Structure: The goal was also to find a compromise between having a lot of structure and no structure at all in terms of requirements engineering processes. Agile software practices do not promote as extensive documentation as the more plan-driven approach. Thus, despite Ericsson moving from the waterfall model towards more agile ways of working, it was important to still keep some structure, i.e., research potential features and write initial high-level requirements before the development started. This was illustrated in the following interviewee's remarks,

> "...it's a compromise. We try to take the best from what is needed, ideas of structure and so on from the UML tools, the expensive ones. And at the same time, but not too much, we make you keep it lightweight." - Interviewee 7

Master Branch Release Ready: By storing the requirements alongside the source code, it facilitates keeping them in sync, and the expectation is thus that the requirements will be up-to-date in every release. Therefore, a goal that Ericsson had in terms of their development was to have the master branch always release ready. This would allow the possibility to look through the history of all the releases and pick any build and release it to the customer. Unfortunately, this is not a possibility for Ericsson today as the current test suite has many expensive tests which take a long time to run. As it is not feasible to run the tests for every commit that is pushed to master, Ericsson has instead opted for monthly or bimonthly releases. However, it is important to note that having the master branch release ready is still an end goal of Ericsson and, thus, an important consideration that the new tool should not be a hindrance in achieving said goal. Thus, the tool is not at fault in this case, and the hope is to reduce the length of the release cycles as quality permits in the future.

Goals & Expectations	Fulfillment
Usability	
Scale-Out	
Automated Test Reporting	
Facilitate Collaboration	
Retain the Structure	
Master Branch Release Ready	

Table 5.1: Table of expectations and whether they have been fulfilled, indicated with green if they have and yellow if only partially fulfilled

5.1.3 Fulfilled Expectations

High experience: The consensus among interviewees with knowledge of the implementation decisions and original goals of T-Reqs was that the tool had achieved what it set out to achieve for the most part. This was emphasized by several interviewees stating they had noticed an increase in requirements updates done by the teams themselves, the scale-out of the company was successful, i.e., they had managed to increase the number of development teams, and the tool was currently being adopted by other organizations across the company. Another interviewee noted that the ease of use was facilitated as the tool was nicely integrated with Git, and the Gerrit reviews also factored into that, i.e., seeing the diffs between commits.

Low experience: For interviewees with less knowledge of the previous requirements engineering tools and thus the decision-making process behind T-Reqs had, for the most part, no specific preconceptions or personal expectations when it came to the tool. This was illustrated by one interviewee when they stated that,

> "...it's like, you know you're a child when you come into the world and it just exists. You don't really question things." - Interviewee 5

The interviewees that expressed any specific personal preconception entailed quite general remarks on the tool. One interviewee remarked,

> "I have only expected it to be kind of a, maybe advanced document for storing requirements definitions. So you would have ID and then you would have some describing text, maybe some attributes. And I think T-Reqs pretty much, yeah, achieves that." - Interviewee 6

Interviewee 12 expressed that they expected the tool to help organize the information about their products, history, and behavior, as well as the reasoning behind decisions. They, however, thought it would be more user-friendly than it is, which in part was due to trouble with learning new syntax. This, they said, was expected but not as smooth as it should be. They also remarked that more senior General System Managers are often deterred from using the tool as they often find it hard to see what they have changed in T-Reqs due to a lack of experience with Git.

Conclusion: In Table 5.1, the goals and expectations of the tool mentioned in Section 5.1.2 are displayed along with the indication of whether they have been met or not.

The color green indicates that an expectation has been met, and yellow indicates that, in some aspects, the expectation has been met and, in some aspects, not. As can be seen in the table, all of the expectations have been met to at least some degree. The reason why two of the goals have not been fully achieved is not due to the tool itself being a hindrance. Instead, the tool's usability is not as high for users that are not as familiar with using Git version control, and keeping the master branch always release-ready has not yet been achieved due to an expensive test suite.

5.1.4 Results from the Survey

A subset of the questions posed in the follow-up survey was to verify that some of the goals and expectations, identified in the interviews, had been met. More specifically, the ones relating to the users of the tool rather than its implementation. That is, the usability of the tool and whether it facilitates collaboration. It should be noted that all interviewees answered each question except Question 8, where one participant chose not to respond.

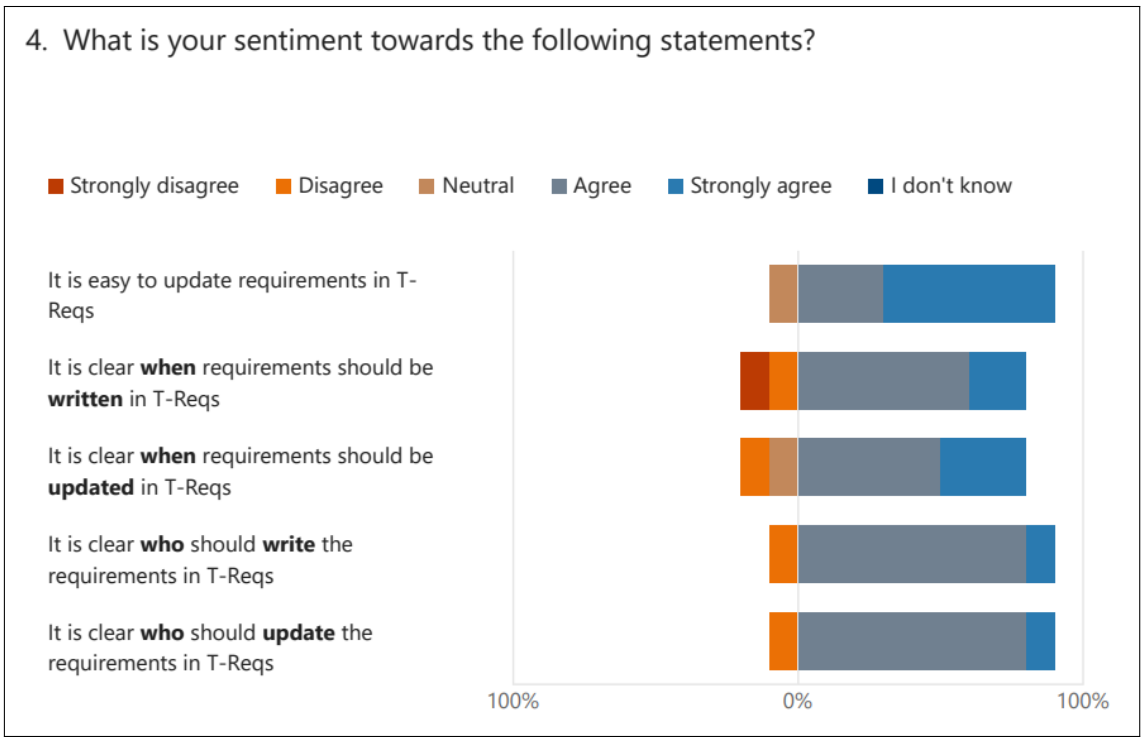

Figure 5.2: Diverging stacked bar chart displaying the responses to Question 4 in the survey

The statements in Figure 5.2 relate to the challenge in large-scale agile development regarding updating requirements, i.e., requirements must be susceptible to change, which is connected to the goal regarding usability, as illustrated in Section **??**. The results show that all except one respondent considered it easy to update requirements in T-Reqs, with the majority, i.e., six out of ten, strongly agreeing with the statement. With regard to when requirements should be written or updated, the majority found it to be clear. Despite this, *Respondent 5* disagreed with both state-

ments, considering it not to be clear. Similarly, *Respondent 1*, strongly disagreed in regard to it being clear when requirements are written but remained neutral on the clarity of when they should be updated. Regarding whether or not it is apparent who should write or update the requirements, the vast majority also agreed, with only one participant (*Respondent 1*), disagreeing with both statements.

In Figure 5.3, participants were asked to give their sentiment regarding statements connected to the goal relating to usability and how easy it is to distinguish when a requirement was implemented or the implementation status of it in a specific release. The respondents' opinions regarding whether it is easy to search for requirements in T-Reqs are a bit divided, with three respondents either disagreeing or strongly disagreeing and six respondents either choosing to agree or strongly agree. One respondent remained neutral. For the sentiment of the respondents regarding the second statement, regarding how easy it is to find in which release a requirement was implemented, one disagreed, four chose to remain neutral, one strongly agreed, and the rest agreed. The responses to the last statement, i.e., whether it was easy to see which requirements are supported in a certain release, with four respondents choosing strongly agree and three agreeing to the statement. However, two participants selected the option that they did not know, and one remained neutral to the statement.

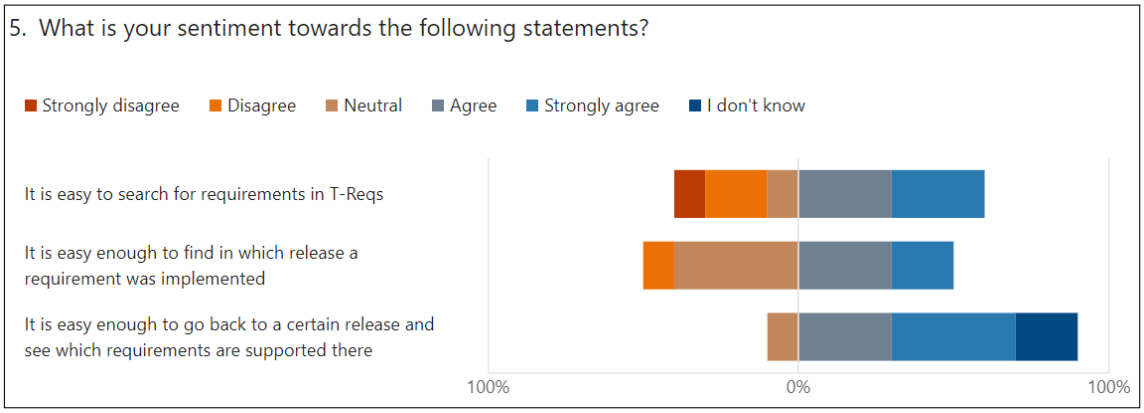

Figure 5.3: Diverging stacked bar chart displaying the responses to Question 5 in the survey

The challenge of consistent requirements quality found in large-scale agile development, connected to the goals of usability and collaboration, was explored by asking for the sentiment of the participants towards the statements seen in Figure 5.4. Regarding the statement whether it is a challenge that requirements reside in different repositories, the respondents were split, i.e., five agreed and four disagreed with the statement, with, however, one person being neutral to the statement. The majority of the participants agreed with the statement that it was hard to keep the quality of requirements in their product domain consistent with the requirements in other product domains. That is, six respondents agreed with the statement, and one respondent strongly agreed. However, two persons disagreed, i.e., considering it not to be hard, and one person was neutral to the statement. Similarly, for the statement regarding whether requirements reviews helped with keeping the quality of the requirements consistent, all except one participant agreed. The respondent that

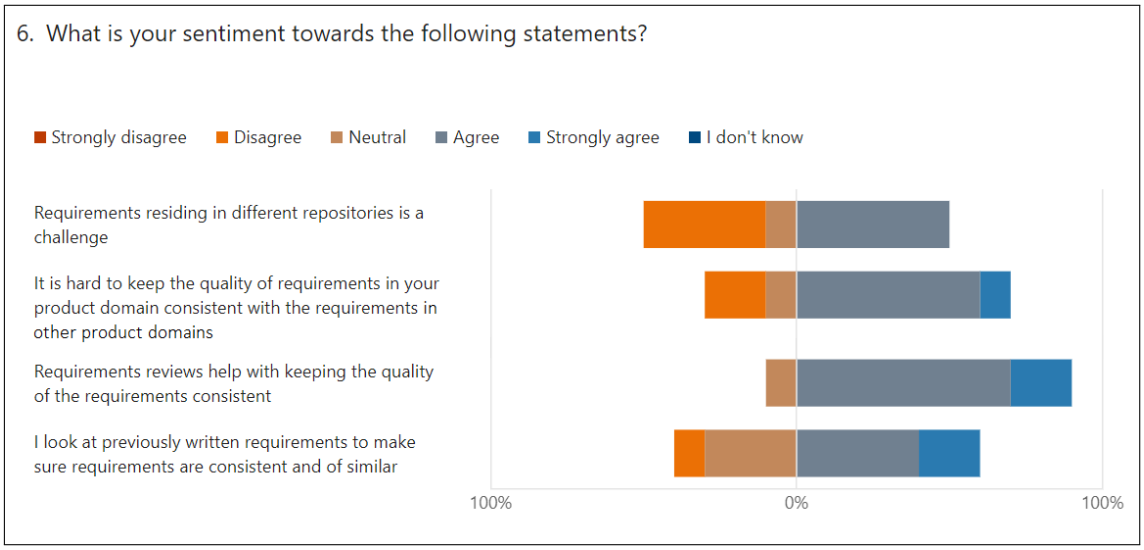

Figure 5.4: Diverging stacked bar chart displaying the responses to Question 6 in the survey

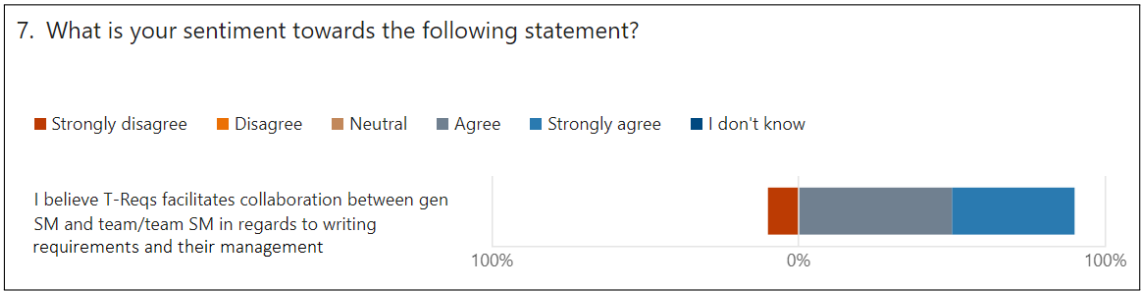

Figure 5.5: Diverging stacked bar chart displaying the responses to Question 7 in the survey

did not agree selected the option to stay neutral. In regard to looking at previously written requirements to make sure requirements are consistent and of similar quality, the majority agreed with the statement, i.e., four agreed and two strongly agreed. However, three respondents remained neutral, and one respondent disagreed.

As mentioned in Section 5.1.2.2 above, one of the main goals and expectations of the tool was to facilitate collaboration. In Figure 5.5, it is illustrated that the majority of the respondents perceive that the tool does facilitate this collaboration, i.e., between the General System Managers and the Team System Managers in terms of writing and managing requirements. That is, five respondents agreed with the statement, and four strongly agreed. However, one person strongly disagreed with this statement.

We asked the participants of the survey to give their reasoning behind their sentiment to the statement in Question 7. From the statements given by the respondents, it is clear that the main collaboration takes place when reviewing requirements, both when they are initially written and then when they need to be updated. *Respondent 2* explained that they considered it to facilitate that something has been agreed upon. However, they added that it is another step that the General System Managers have to take on, i.e., reviewing requirements updates, and takes more time than a meeting. Nevertheless, this is more due to the General System Managers being overloaded in regard to tasks than the tool itself. Similar remarks were given by other respondents, i.e., emphasizing that the tool was not a hindrance but rather the willingness of employees to be involved with the tool. This was further illustrated by *Respondent 3*, as they remarked that the tool was good at facilitating this collaboration as long as effort was being made by both parties to view changes and perform reviews. Additionally, the respondent stated that a current issue was slowness in regard to feedback, as all requirements need to be reviewed and approved by General System Managers. *Respondent 6* elaborated on this collaboration, i.e., in terms of reviewing requirements, by stating that the General System Managers can ensure that the quality and structure of the writing are maintained and consistent. The Team System Managers can then offer comments regarding the technical accuracy of the written requirements. From another angle, *Respondent 4* considered the collaboration to be made easy in the repository by using Git and Gerrit. The reasoning that *Respondent 1* gave in regard to strongly disagreeing with the statement was due to the current state of the requirements, which they believed were misaligned with the products. They further stated that Ericsson should have a new way of handling requirements.

The statements in Figure 5.6 relate again to the goal of the usability of the tool, more specifically, that the tool should be both easy to set up and use. Half of the participants strongly agreed that the tool was easy to set up, two remained neutral, and the rest agreed. Four respondents agreed that it was easy to use the plugin in the code editors (T-Reqs editor in Figure 5.6), and three strongly agreed with the statement. However, three respondents chose the option that they did not know. In regards to the usability of T-Reqs Web, all except one thought that it was easy to use, with two respondents strongly agreeing with the statement. Of the different views offered for T-Reqs, T-Reqs Viewer seems to be the one most difficult to use, as three participants disagreed with the statement that it was easy to use,

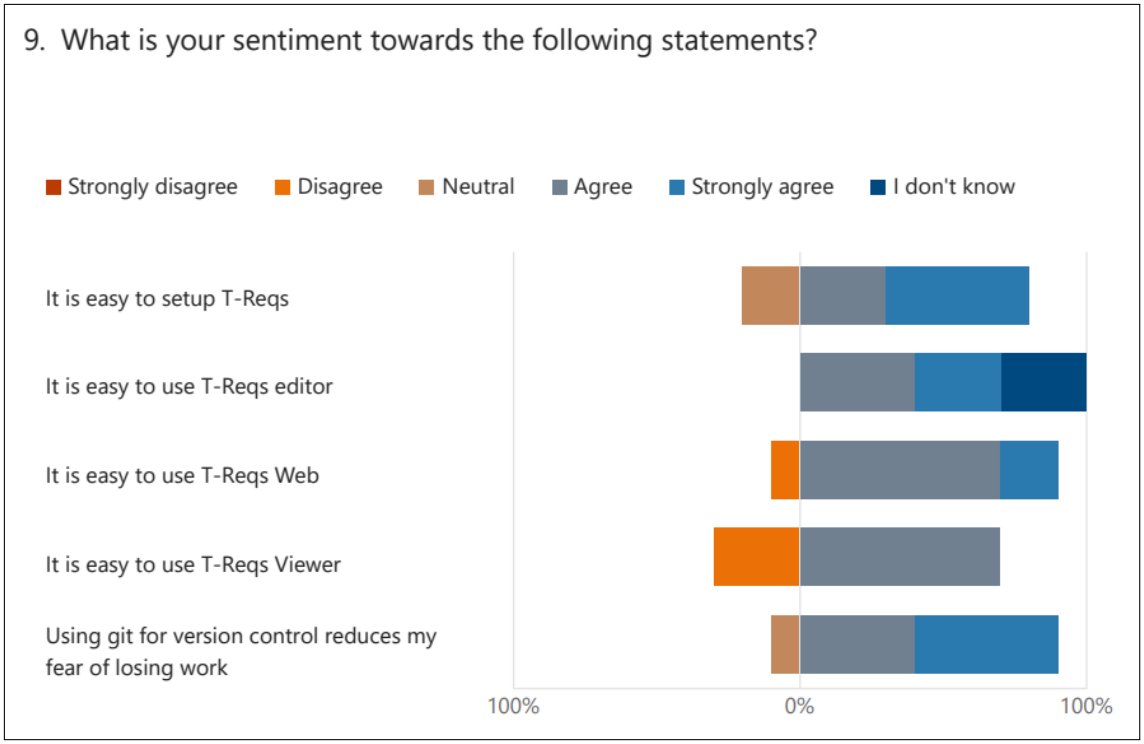

Figure 5.6: Diverging stacked bar chart displaying the responses to Question 9 in the survey

but the rest agreed that it was easy. Regarding the expectation that Git version control would reduce the fear of losing the work that has been done, the statement seemed to resonate with the respondents as half strongly agreed with the statement, four agreed, and one participant remained neutral.

In conclusion, the survey results showed that the tool's usability was good, with only some disagreeing with the search for requirements in T-Reqs being easy and using the T-Reqs Viewer. This corresponds, however, with the fact that six of the participants strongly agreed that they were comfortable with using Git version control, and the other four agreed with the statement. That is, the population of employees not as comfortable with Git is not represented in the survey answers, which were one group considered to have the most issue with the usability of the tool according to the interviewees. The results confirm that collaboration is facilitated by the tool, which also helps with keeping the quality of the requirements consistent. The challenge, however, is that requirements reside in different repositories and that it is hard to keep the quality between them consistent. One outlier considered the current state of the requirements not adequate and called for a new way of handling requirements.

5.2 RQ2 (Advantages and Disadvantages)

What advantages and disadvantages of T-Reqs can be observed in practice in relation to traditional requirement tools?

The results show that developers perceive the main advantage of T-Reqs to be its integration with the development environment. By storing requirements as text files in Git along with source code and test cases, the tool allows for easy access, updates, tracing, version control, and baselining. Requirements can also be seamlessly integrated into the review process, thereby promoting collaboration and usability. From a broader organizational perspective, the main advantage of T-Reqs for Ericsson is its cost-effectiveness as a solution developed in-house. However, the research findings indicate that infrequent users lacking familiarity with standard IT technology may encounter significant usability challenges when using T-Reqs. In addition, due to the high freedom of the tool, the absence of clear guidelines can affect the quality of the requirements, highlighting the need for effective training and support for the tool's users.

To answer this question, we present the advantages and disadvantages of T-Reqs, Rational RequisitePro, and Rhapsody based on the themes identified from the semi-structured interviews. A summary of all advantages and disadvantages of T-Reqs found and presented in this chapter are provided in Table 5.2. Lastly, we emphasize the practical aspects of T-Reqs' advantages and disadvantages in Section 5.2.5 that discusses observations of T-Reqs in use, addressing contextual factors that affect its performance.

5.2.1 Usability

Integrated into the Development Environment: T-Reqs has the advantage of being integrated with the environment used for source code and testing, i.e., in a version control system as text files. This allows developers and testers to have their daily tasks in the same place without needing to switch to a separate program for managing requirements. In the words of one interviewee,

> "...everything located in one place [...]. As a developer it's nice to be able to use what you are usually working with the other day to day tasks" - Interviewee 13

However, this adds an extra step in the work process for most General System Managers, who are not usually working in the development environment. They must then connect to an external virtual machine for the requirement handling in T-Reqs, which can take some time. Similarly, Rational RequisitePro and Rhapsody faced integration challenges within the software environment due to their lack of support on the Ericsson platform. Consequently, virtual machines were employed to execute these programs, resulting in prolonged startup times and a cumbersome user experience.

	ADVANTAGES	MIXED ADVANTAGES/DISADVANTAGES	DISADVANTAGES
5.2.1 USABILITY	• Can dynamically render an HTML preview of the document • Easy access to information regarding the requirements	• Integrated in the development environment • Built upon standard IT technology (e.g. Git, Linux) • Has a high degree of freedom (low restriction of what to write in a requirement)	• Tool setup • Filling out tags and attributes • Creating PlantUML layouts • Lack of visualization of how requirements are connected • Lack of guidelines regarding what to write in the requirements
5.2.2 IN-HOUSE DEVELOPMENT VERSUS LICENSED TOOLS	• Lower cost than licensing tools • Improvements of the tool via change requests is possible due to in-house development • No unnecessary features • Allows multiple users to have access to the requirements (as opposed to limited amount of licenses)		• Missing functionality • New functionality/bug fixes take a long time to implement
5.2.3 STORAGE & VERSION CONTROL	• Easy synchronization with code and test cases (keeping requirements release ready) • Easy tracing • Easy version control • Easy baselining • Enables automated test reporting • Easy to simultaneously update requirements/-source code/test cases • Reduced fear of making mistakes/losing work		• Solving merge conflicts in Git • Multiple repositories (mapping & different WoW)
5.2.4 COLLABORATION & ACCESSIBILITY	• Allows multiple users to work on the same requirements artefact simultaneously • Change tracking in Git (who changed what) • Integration with review process (Gerrit)		• Lack of guidelines regarding who should write the initial requirements

Table 5.2: Advantages, disadvantages, and mixed advantages/disadvantages of T-Reqs identified through interviews with Ericsson personnel.

Reuse of Standard IT Technology: T-Reqs, by leveraging standard IT technologies such as Markdown files, PlantUML, Git, and the Linux OS, aligns well with the prior experience of many users, particularly developers, thereby enhancing its usability. As highlighted by one participant,

> "...what I liked about T-Reqs was that it was based around these markdown files, right? So I have some experience with markdown before, so I thought that was nice like it was in the format I could understand." - Interviewee 5

Similarly, another interviewee reiterated the perceived benefit.

> "The thing is that I was kind of familiar with, because for T-Reqs you need to use Git and Gerrit and stuff. And I was thankful for that because I was working this Git and Gerrit in my previous job." - Interviewee 10

The stripped-down feature set, coupled with the standard technologies, keeps the tool's interface simple and direct - primarily a terminal and a text editor - facilitating easy reading and writing of requirements. However, some interviewees expressed some initial struggles with the syntax of requirements.

The use of standard IT technologies also eases the onboarding process for new employees, contrasting with standalone programs like Rational RequisitePro and Rhapsody, which require specific training. However, while these features are convenient and user-friendly for developers, they may be challenging for those less familiar with these technologies. Users who face a technical gap can find aspects of the tool, such as resolving Git merge conflicts, daunting and potentially off-putting. Despite a few of the interviewees reporting problems with usage, there is recognition that the terminal use might be excessively technical for some users. While resistance to it has decreased over time, some organizations still exhibit reluctance towards its adoption. A General System Manager participating in the interviews highlighted the challenges they face, where even minor modifications can be time-consuming. This is primarily attributed to the need to recall the specific command-line instructions, leading to frustration as they tend to forget the process when there are significant intervals, sometimes spanning up to three months, between tool usage.

Interviewee 13 elucidated an important aspect of tool selection for requirement management, emphasizing the correlation between the proximity of teams to requirement responsibilities and the choice of tools. They suggested that when teams are closely involved with the requirements, employing them in a version control system repository is advantageous. However, for users such as General System Managers, who do not typically operate within the development environment, a tool with a Graphical User Interface (GUI) could provide a more user-friendly experience, easing the management and manipulation of requirements. This distinction underscores the importance of considering the user's context and familiarity with the development environment when selecting requirement management tools.

Visualization: As a supplement to the textual requirements in T-Reqs, models can be created. T-Reqs supports Model Driven Development by enabling the creation of UML diagrams, providing a visual representation of various use cases. To achieve these diagrams, PlantUML is used, which has the big advantage of enabling version control and easy updates of the diagrams due to it being text-based and has been

described as a powerful tool. On the other hand, a few downsides to PlantUML have been identified as a theme among the interviewee participants. As stated by the interviewees, it can be extremely frustrating to work with as the layout created is not predictable, and the coding structure is quite rigid. A common way to mitigate this is, as described by the interviewees, to copy old diagrams and edit them to fit the new use case. It is also difficult to design the diagrams to be descriptive and easy to read, which does not scale well with complexity. This is unlike the models in Rhapsody, which have been described by the interviewees as being easy to use. Additionally, there is no effective way to include images in T-Reqs because they lack the benefit of version control, making it challenging to compare changes easily. This limitation aligns with the shortcomings observed in Rhapsody models.

Multiple participants stated that having an easier modeling interface is not crucial or a hindrance and emphasized that textual requirements are easier to read and update. However, the interviewees did remark that an easier interface for PlantUML would be beneficial.

Furthermore, T-Reqs lacks in providing visual representations that illustrate connections between requirements, which could be a consideration for certain users or use cases, especially when working on umbrella features.

Tool setup: Each of the three tools referenced in this study, i.e., Rational RequisitePro, Rhapsody, and T-Reqs, was said to share a common challenge: a cumbersome initial setup. The process of installing and integrating the tools within the existing systems and workflows can be complex and time-consuming. There may be technical challenges or incompatibilities that need to be addressed, requiring expertise in both the tool itself and the wider system architecture.

Writing and Reading Requirements: T-Reqs has received praise for its ease of use and accessibility of information by some of the interviewees, particularly through its web interface that provides an easily readable view of the requirements. One interviewee explained their use of T-Reqs Web, specifically when they had a lot of requirements to read through,

> "...it's easier to do it on the HTML version. So I use that one a lot." - Interviewee 9

The search functionality in the web interface has been praised by some; but others find it less attractive. *Interviewee 13* described it as slow to load documents as it searches the entire repository, even in cases where the user knows which document the desired requirement is in. It was also brought up by the same interviewee that pop-ups appear and obstruct the browser window when searching for requirements. *Interviewee 14* also wants the search to show the implementation and not only the requirement when looking for test cases, as that is more relevant to their use case. Because of these reasons, some prefer the ability to search using commands in the repository itself, but that also has its own limitations.

The tag system, i.e., the architectural way of storing requirement artifacts (refer to Section 3.2), its attributes, and the system for validation of tags are useful assets of T-Reqs and help to keep track of the development and its connection to the requirements. However, some users have expressed frustration with the need to fill in T-Reqs attributes in the tags, as it can be a time-consuming task. Rational

RequisitePro faced this issue as well and required even more information to be filled in towards the end of its life cycle, including information that was not even known by the users. In addition to the tags, other assisting information, e.g., PlantUML models enabled by T-Reqs, has been mentioned as valuable.

As mentioned above, T-Reqs uses Markdown files to document requirements, making it easy for those with prior experience and challenging for those without it. Having a continuous dynamic rendering of the Markdown to HTML in a separate window mitigates some of the difficulties of not writing in plain text and has been mentioned as particularly good for the tool's ease of use. Rational RequisitePro, on the other hand, utilizes a Word interface where no rendering is needed, which makes it easy to read and write requirements. Rhapsody's interface was reportedly easy to use and allowed users to achieve desired modeling layouts with ease, which is not the perceived case in T-Reqs, as PlantUML in T-Reqs poses challenges in achieving desired layouts. Some aspects were more difficult in the Rhapsody interface, particularly when performing tasks such as copying and pasting diagrams. Due to the visual representation that a modeling tool offers, it was easy to understand the requirements in Rhapsody.

General Positivity: Low-experienced users of requirements engineering tools were especially positive about the use of T-Reqs. T-Reqs's core system has been described as highly effective, and it facilitates efficient writing. One interviewee even expressed uncertainty when asked about the potential drawbacks of T-Reqs, stating,

> "I'm trying to think of something which is not good about it, but I'm not sure I can think of anything. I guess there's the thing that it might be tricky for people who are not used to code or terminal, things like that." - Interviewee 3

Interviewee 5 shared a similar sentiment of conflict regarding potential disadvantages, saying that the tool is fine, but the ways of working and surrounding artifacts are the disadvantages. However, they acknowledged that there were areas for improvement within T-Reqs itself. High-experienced users were more likely to say that T-Reqs is not as good in terms of usability compared to RequisitePro and Rhapsody. This was mainly due to more steps needed to complete a task, since all of the high-experience users are General System Managers and therefore have to connect to the development environment. The Markdown formatting and the lack of a Graphical User Interface were also mentioned as barriers to ease of use.

Balancing Flexibility and Structure in Requirement Documentation: Rational RequisitePro and Rhapsody offer structured documentation formats which have been likened to a labyrinth by an interviewee. The advantage of these structured tools is their ability to guide users and improve the quality of requirements by dictating when and how things are written and what metadata is needed. However, this rigid approach can also prove frustrating and time-consuming for users, limiting their freedom.

Contrastingly, T-Reqs offers a more flexible, free-form approach to writing requirements, which has not shown these problems. This high degree of freedom, however, comes with its own potential risks. As *Interviewee 7* cautioned, users must be aware of the potential dangers associated with such flexibility, with the

most prominent danger being inconsistencies in the quality of requirements, which will be discussed further in Section 5.2.5. Despite this, many users prefer the freedom and sense of empowerment T-Reqs provides, as well as its accessibility and support for agile work processes.

The introduction of a tool like T-Reqs, which provides a high degree of freedom, must be accompanied by proper measures to ensure its effective use. These measures include proper training and guidance for users, clear guidelines and standards for documentation, and a consistent, organized approach to managing requirements. Without these precautions, the flexibility provided by T-Reqs could lead to misuse and potential risks, undermining the tool's advantages and hindering its benefits. Thus, it is a delicate balance between providing the freedom desired by users and ensuring the quality of requirements documentation.

5.2.2 In-House Development Versus Licensed Tool

Ericsson developed T-Reqs in-house instead of buying a license for a tool, as had been the case for Rational RequisitePro and Rhapsody previously. This decision comes with its own set of advantages and disadvantages.

Cost: When Rational RequisitePro and Rhapsody were in use, the tools' fees were a significant disadvantage for Ericsson. Even though these tools were costly, some integration was still needed within the environment they were to operate in, which incurred an extra cost for Ericsson. The tools also required user training, as they were separate programs, often with proprietary technologies unfamiliar to the users. The companies providing the services also wanted customers to become dependent on their tools. This is illustrated by the difficulties of migration from Rhapsody, and there are still abnormal artifacts in T-Reqs from the poor migration.

Change Requests: With an in-house developed tool like T-Reqs, users are closer to the development process of the tool. T-Reqs users can request changes or additional features to be implemented by the T-Reqs maintenance team at Ericsson. Conversely, such a request would unlikely be implemented within a licensed tool. T-Reqs is continuously being developed, and shortcomings are seen as temporary. The only remarked downside is that it can take a long time for proposed changes to be implemented, with some taking longer than others, depending on whether multiple people request the same change or not,

> "Because if I'm the only one who's, you know, that's screaming about the problem, and nobody else is, then they won't prioritize it. So that takes quite a long time." - Interviewee 6

On the other hand, several other interviewees expressed positive experiences, emphasizing that they have received prompt responses and feedback from the T-Reqs development team.

Functionalities & Features: One advantage of licensed tools is that they often come with a wider range of features and fewer bugs right from the outset. T-Reqs currently lacks certain features that are commonly found in requirements engineering tools,

such as the ability to display syntax errors and a requirement history function. At the same time, Rational RequisitePro received criticism for having a large number of features that were not commonly used. One participant expressed this sentiment, remarking that,

> "...we only want to use 5% of what they have put in there..." - Interviewee 7

Despite the large feature set, some desired features were missing in Rational RequisitePro, e.g., the ability to change multiple requirements at once. One interviewee expressed their desire for a proprietary tool in terms of the increased quantity and quality of documentation that would be available in comparison to a tool developed in-house.

Interviewee 9 highlights that Ericsson is a software company with previous experience in developing their own tools and has a special organization for it. This made it beneficial for them to develop T-Reqs, as nothing on the market was a good fit for their requirements engineering needs. Additionally, the interviewee expressed that T-Reqs might not be the best at everything, but it handles the requirements fine, and because it is developed in-house, it is tailored to their needs.

5.2.3 Storage & Version control

Integrated with CI processes at Ericsson: Storing T-Reqs requirements and their metadata in text files, as well as source code and test files in a version control system, offers several advantages in addition to the usability aspects discussed, that is being integrated into the development environment and using standard IT technology that is familiar to the employees, here the version control system Git. It also allows for easy synchronization between requirements and source code, i.e., having the requirements up to date in every release. Additionally, baselining, version control, tracing, and automated test reporting become simple tasks. Updating the source code, test cases, and requirements simultaneously is also possible with this approach. Searching for requirements is also easier as they reside in the repository as text files. The files themselves are also smaller in size compared to Rational RequisitePro and Rhapsody files, which saves disc space. Moreover, Git, in combination with Gerrit, offers collaborative features such as peer reviewing and change tracking, which allow users to see who has modified what. All in all, T-Reqs is well integrated into the CI machinery at Ericsson.

Tracing: Tracing is perceived as one of the most prominent advantages of the tool among the interviewees, both to keep an up-to-date version of the requirements and to ensure the coverage of the test cases. It has been described as well-implemented and is a good reason to use the tool. One thing that is missing in regard to the tracing is feedback on broken trace links. If a tag is removed from a test case or the test case is removed, there is no feedback to the users, which may result in lost coverage. A proposed solution would be to add warnings in Gerrit when this occurs.

Version Control in Previous Tools: Rhapsody artifacts were stored similarly with the source code but could not take advantage of version control in the same way due to the complexity of the information stored. The requirements were stored as flow

charts, i.e., text files with extra information which is not human-readable text. They could, therefore, not be version-controlled or compared like text files and PlantUML diagrams, meaning version handling had to be executed separately and manually. Due to Rational RequisitePro storing requirements in a database with Word files, comparisons could be made quite easily, however, the version control required manual labor. The requirements also had a poor connection to the source code and tests since they were stored in different places and ways. Baselining was also performed separately from the tests.

Added Level of Confidence: The use of Git as a version control system has the advantage of providing an added level of confidence and reducing the fear of making mistakes. Unlike previous tools, Git allows for undoing actions and reverting to a previous version of the code without the need to go back manually and undo your mistake. However, using Git may also lead to merge conflicts, which can be challenging for inexperienced users to resolve.

Multiple Repositories: The existence of multiple repositories within Ericsson, with varying levels of requirements in them, has presented challenges in achieving complete mapping between the repositories. Despite this, *Interviewee 8* has indicated that the need for full mapping is not essential, stating that

> "...we don't really have the need for having 100% mapping everywhere." - Interviewee 8

However, usability issues have been reported when transitioning between repositories, as they operate using different ways of working, resulting in a lack of consistency in the user experience.

5.2.4 Collaboration & Accessibility

T-Reqs is a tool designed to facilitate collaboration and enhance efficiency in the realm of requirements management. One of its main advantages is the ease with which requirements can be peer-reviewed, akin to code review, owing to its storage method and the accessibility it provides to all team members. This widespread accessibility ensures that development teams can readily familiarize themselves with product specifications, resulting in more streamlined development processes. At Ericsson, the tool used for peer review is Gerrit which integrates nicely with Git. A reviewing tool like Gerrit, which is integrated into the normal ways of working of the users, has been deemed necessary for T-Reqs's success. *Interviewee 14* stated that the tool facilitates discussion about requirements between the team and the General System Manager. They would then bring up T-Reqs Web to show and discuss requirements. Another feature praised for the same reason is the ability to link requirements in e-mails which simplifies the discussion about requirements in a distributed workplace.

The transparency afforded by T-Reqs enables users to track changes and identify the responsible party, thereby supporting agile methodologies. Furthermore, testers can work directly within the repository to update traces between tests and requirements. Utilizing a version control system for document handling allows mul-

tiple users to collaborate on the same document simultaneously, eliminating the need to reserve and check out documents for edit.

T-Reqs has been described by the interviewees as the best tool for collaboration when compared to Rational RequisitePro and Rhapsody, mostly due to the review cycle in Gerrit, which is similar to normal code reviews. When Rational RequisitePro was used, they sent Word documents in e-mails describing the updates, but nowadays, it would be somewhat easier with modern Word collaboration features.

In summary, T-Reqs offers a robust platform for efficient requirements management that promotes collaboration, transparency, and accessibility. This sets it apart from its predecessors, which had a limited number of licenses and, therefore, a limited number of users with access to the tool.

5.2.5 Practical Observations of T-Reqs in Use

Some aspects observed from the interviewees may not be directly linked to T-Reqs but affect its performance. This is mostly connected to the ways of working and culture at Ericsson.

Collaboration Between the General System Manager & the Team System Manager: Before the implementation of T-Reqs, Ericsson had identified the General System Managers as a bottleneck in product development, and with T-Reqs, they sought to bring requirements closer to the team to alleviate some of the workloads. Much pressure still lies on the General System Managers, and many do not want to spend time learning a new requirements engineering tool. As mentioned earlier, the required competence to quickly get into T-Reqs may also be lacking, especially for General System Managers who have not recently held a developer role. Due to the General System Managers' lack of time and, thereby, competence, requirements management is more or less exercised by the team, specifically by the Team System Manager, who has the role of updating requirements in T-Reqs. The problem that has arisen is that there are inconsistencies in what is passed down from the General System Manager to the Team System Manager, as highlighted in the following statement by *Interviewee 3*,

> "I thought it would be like we would have some like high level requirements to start with and then maybe we, me as a team SM with the team would kind of make more detailed requirements based on that. But usually when we started a feature there was nothing and so we had to kind of write it from scratch" - Interviewee 3

As opposed to the statement above, other Team System Managers expressed that they had General System Managers that gave them high-level requirements to start with or a requirements structure to fill in. The level of detail and structure provided by General System Managers to their teams varies depending on the individual and the context. One participant that is currently working as a Team System Manager described the requirement handling as a struggle due to the lack of guidance from the General System Manager. Several interviewees expressed that the ideal would be for the General System Manager to hand down high-level requirements to the Team System Manager and the team, but they recognize that the General System Managers are too busy. Therefore there is an overarching theme

among the participants that handing down the feature study (PowerPoint) is good enough. Some participants even say that it is the desired way of working as handing down high-level requirements in T-Reqs leads to bloating as they will unnecessarily reside in the repository.

The transfer of information from the General System Manager to the Team System Manager lacks clear guidelines, leading to potential issues. This was illustrated in the following remark,

> "It's easy to think that somebody else should do it. We can leave this to the team. And then the team think, ah, we got too few, too little inputs for our work. Why didn't you do this before?" - Interviewee 2

Regardless of who takes the lead, the requirements must be in place prior to development, which has been proven to not be the most common outcome as multiple interviewees narrate their experiences where requirements are pushed later in the feature development. Ultimately, it is the responsibility of the General System Manager to ensure the quality of requirements. Failure to collaborate effectively in this area can result in scope cuts, highlighting the importance of a clear understanding and process for information transfer between the General System Manager and the team.

Quality: When asked about what impacts the quality of requirements more of the processes or the tool, *Interviewee 5* emphasized the importance of processes and the competence of individuals involved. This sentiment was reiterated by several interviewees, mentioning the importance of guidelines and how the tool is a clean slate, especially tools with a high degree of freedom. As requirements in T-Reqs are written in a free form in a text file, anything can be written, and as guidelines regarding how to define a requirement have been lacking to guide the users, the quality of the requirements has not been consistent. For example, one employee expressed how it was challenging to know how much to write, and initially, they wrote too much in each requirement. They had later been told that each requirement should be singly validated by a test case since if the requirement is only partly covered by a test case, it leads to ambiguity and validation problems. In another case, it was mentioned that the guidelines indicate that some requirements need to be in place before the development starts, but not the level of detail needed. In the past, the inconsistencies of the level of detail written in requirements were also proven to be a problem as it led to chaotic management and resulted in people not wanting to update requirements in T-Reqs. It is said this was due to an organizational restructure and ramp-up during a new product development where a lot of new teams were formed. The teams got more responsibility for the requirements but did not have the right competencies to manage them. In Rhapsody, a similar occurrence took place where requirements were too detailed and got too hard to maintain. Together with the low pressure of updating requirements from the organization, people stopped updating requirements, and requirement debt built up. After an incident where a product failure happened due to this, more pressure was put on proper requirements handling.

How to specify a child requirement, what to put in the attributes, and how to write good requirements have also been unclear. Thus far, the de facto approach has

been to emulate older requirements or ask other employees to upkeep the quality of requirements. Even the General System Managers do not always know the latest way of working or have misconceptions themselves, which has led to contradictory answers. Guidelines and templates are being formed at Ericsson to show clarity and completeness regarding writing requirements and also support the process of reviewing requirements. It is emphasized strongly by multiple participants that clear guidelines and education have to be available for a successful adoption of a requirements engineering tool.

Spending Time on the Requirements: Requirements are not artifacts that are to be delivered to the customer, and therefore they are sometimes overlooked, especially in an agile environment where flexibility and customer collaboration are prioritized over documentation and systematization. Previously, Ericsson used requirements as a contract to the customer of what would be developed, and the description of the product would be written separately after development. This approach was time-consuming and not suitable for a large-scale agile way of working. The development time was being squeezed, and consequently, the requirements were being written as late as possible. In addition, Ericsson lacked structure for systematization, and requirements were being ignored due to tight deadlines, which led to scope cuts and requirement debt. It is expressed by *Interviewee 7* that there has to be a balance for the amount of systematization, which must not be too rigid or too loose. In the end, spending time on requirements early will shorten the development time as developers will know what to write and test while also not creating requirement debt. Currently, Ericsson has opted for a combination where the organization works in accordance with a plan-driven model, but the development teams work agile. This combination seems to cause confusion as participants have expressed a lack of clarity as to when to write requirements. The consensus among the interviewees was that having requirements written before starting the development is easier. However, that is not in accordance with agile practices. T-Reqs is not seen as an obstacle to work agile, but rather it's the way of working, e.g., when to write requirements and how to write good requirements and the competences of the people working with the tool, as expressed by *Interviewee 5*. It was also said by *Interviewee 9* that T-Reqs facilitates an incremental work process due to the seamless integration in the review process, which makes it easier to work agile than other tools used by them. However, *Interviewee 13* stated that the review feedback by the responsible General System Manager needs to be delivered faster to fully work agile with the requirements.

On a team level, opinions differ from team to team if they prefer a more plan-driven or agile approach. Some teams favor establishing a set of early requirements as a formal agreement of what to develop rather than a description of the implementation written post-development. Requirement meetings within the teams have also revealed uncertainty regarding the desired functionalities, thereby underscoring the importance of initiating the requirement writing process early. However, other teams feel restricted by too many requirements and do not deem it necessary to operate with formal requirements present. Team System Managers have expressed their role as the sole team member who emphasizes the requirement management

work within the team, and at times, they perceive a lack of involvement from others who may question the time invested in requirements-related activities. Additionally, the challenge lies in motivating the allocation of time towards requirements since they may not be directly delivered to the customer.

In general, among the interviewees, it is seen as a problem if the requirements are formed too late in the development process. Due to the general outlook on requirements, T-Reqs might not reach its full potential with its ability to accommodate access to the requirements for all members of the team. Despite this, having a limited amount of people responsible for the requirements might be beneficial, as remarked by one interviewee,

> "Too many cooks spoil the broth." - Interviewee 4

5.2.6 Results from the Survey

The survey served as a means to verify the identified advantages and disadvantages of T-Reqs, which were derived from the insights gathered during the interviews with individual subjects. The participants were asked to provide their agreement or disagreement with these identified aspects. The results of this validation process are summarized in Figure 5.7 (advantages) and Figure 5.9 (disadvantages), displaying the level of agreement among the survey respondents regarding the reported advantages and disadvantages of T-Reqs.

Furthermore, the participants were given the opportunity to prioritize these aspects by selecting the main advantages and disadvantages of T-Reqs. Figures 5.8 (advantages) and 5.10 (disadvantages) present the prioritized topics based on the participants' selections, offering a clearer understanding of the key perceived advantages and disadvantages of T-Reqs according to the survey respondents. All participants of the survey answered every question presented below.

The survey results revealed strong support for several advantages of T-Reqs. Firstly, participants agreed that it is beneficial for them that T-Reqs is *integrated into the development environment*, indicating that the tool seamlessly integrates with their existing workflow. One outlier (*Respondent 1*) disagreed that it is an advantage for them. Similarly, respondents strongly agreed with the advantage of T-Reqs being *reliant upon standard IT technology*, such as Git and Linux, which further contributes to its compatibility and ease of adoption. Again one outlier (*Respondent 1*) remained neutral to the choice.

Participants also expressed agreement that T-Reqs *avoids unnecessary features*, emphasizing its focus on providing essential functionalities without unnecessary complexity. Furthermore, respondents recognized T-Reqs' ability to *dynamically render an HTML preview of the Markdown document*, enabling convenient and real-time visualization. The majority, i.e., seven respondents, strongly agreed to this statement, while two respondents expressed agreement and one maintained a neutral stance.

The survey results also highlighted the advantages of T-Reqs in terms of *synchronization with code and test cases, tracing, version control, baselining,* and *automated test reporting*. Participants agreed that T-Reqs facilitates these aspects, enabling efficient management and maintenance of requirements throughout the de-

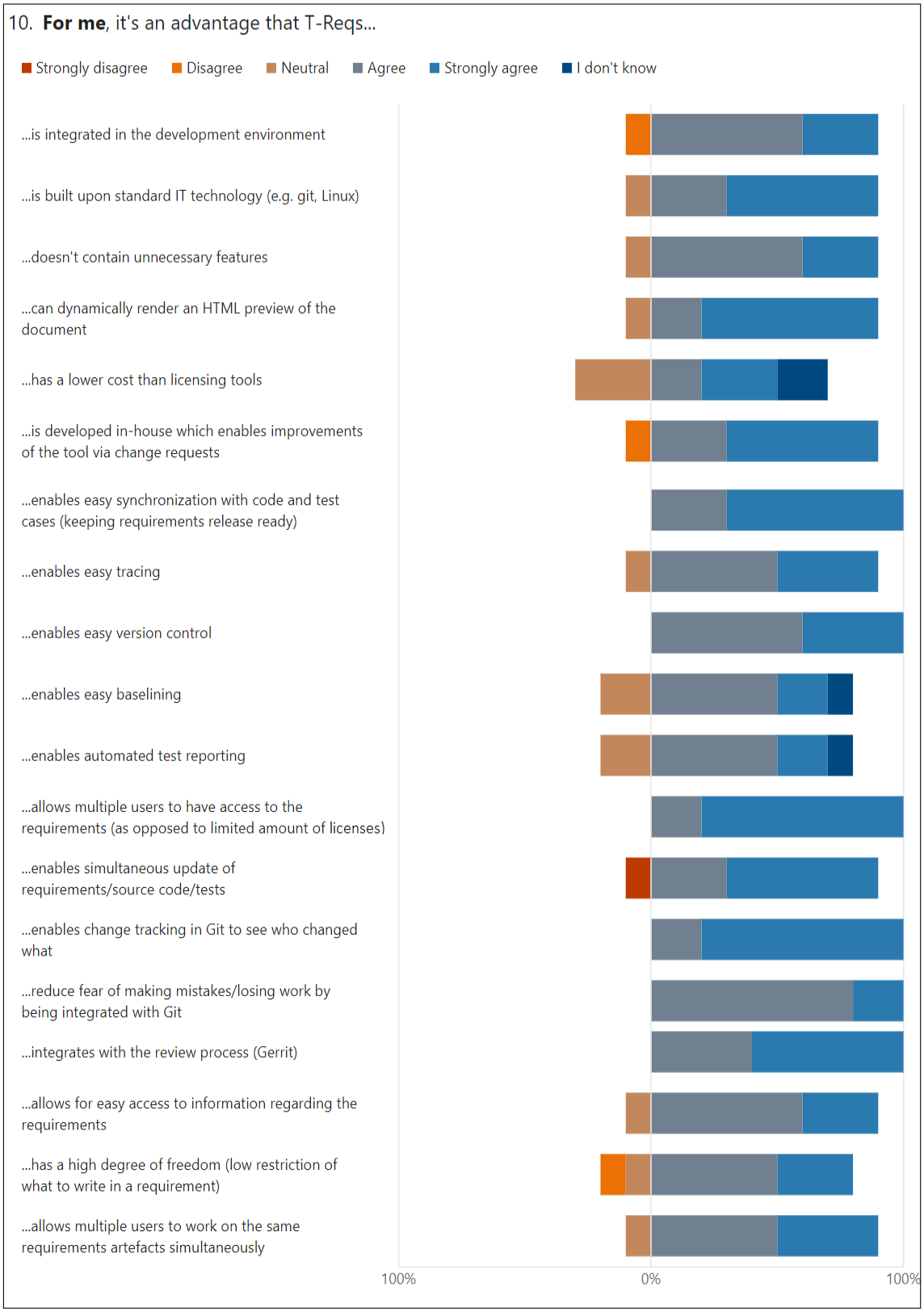

Figure 5.7: Diverging stacked bar chart displaying the responses to Question 10 in the survey related to advantages of T-Reqs

velopment process.

Moreover, respondents appreciate the collaborative nature of T-Reqs, as it *allows multiple users to access the requirements*, as opposed to having a limited number of user licenses. Eight participants strongly agreed with this statement, and the rest agreed. The same distribution was recorded for the advantage of *change tracking in Git* to see who changed what. Being *integrated into the review process* in Gerrit also had positive results, with six persons strongly agreeing to its advantage and three agreeing. Finally, respondents appreciate that T-Reqs allows *multiple users to have simultaneous access to the requirements and work on the same artifacts* with the majority agreeing or strongly agreeing, one person being neutral, i.e., *Respondent 6*.

Regarding *cost*, participants held a neutral stance, with two respondents being unsure of the advantages and the rest being spread across neutral, agree, and strongly agree. However, respondents acknowledged that T-Reqs, being *developed in-house, allows for improvements of the tool via change requests. Respondent 3* did not agree to this being an advantage for them.

The majority of the participants agreed or strongly agreed that T-Reqs' ability to *enable simultaneous updates of requirements, source code, and tests* was a substantial advantage. *Respondent 1* strongly disagreed with this sentiment. Likewise, T-Reqs' *reduced fear of making mistakes or losing work* as a result of integration with Git was viewed favorably by the respondents. All of the participants either agreed or strongly agreed that this feature was an advantage, underscoring the importance of reliability and assurance in tool use to support its usability. *Easy access to information regarding requirements* was another aspect that the majority of respondents agreed upon.

Moreover, T-Reqs' *high degree of freedom*, allowing low restriction on what to write in a requirement, received generally positive responses. While there was one outlier (*Respondent 4*) who disagreed and one who remained neutral (*Respondent 8*), most participants either agreed or strongly agreed that this feature was advantageous, illustrating the appreciation for flexibility in the tool.

By picking the most important features of the perceived advantages, we can conclude a consensus of what the participants consider to be the most important features of the tool. The results are available in Figure 5.8. All participants except one (*Respondent 9*) agreed that the real-time *rendering of the Markdown to HTML* is one of the most important features. The second most picked aspect, with seven respondents, is that T-Reqs *allows multiple users to have access to the requirement* and is not restricted by a limited number of licenses. Subsequently, the following two aspects garnered six selections each: that T-Reqs is *integrated into the development environment* and that it is *integrated into the review process in Gerrit*. This was followed by *easy version control* and *enabling multiple users to work on the same artifacts simultaneously* that each got chosen five times. The aspects that were selected the least were the *lower cost* of the tool compared to the previous tooling and *easy baselining*, indicating that despite the majority considering these aspects as advantages, they are not the most important aspects of the tool.

The survey unveiled several potential concerns among users of T-Reqs, which are presented in Figure 5.9. First, there was a tendency among participants to agree that the *lack of visualization for the connections between requirements* constitutes

Figure 5.8: Multiple choice question from the survey for identifying the main advantages of T-Reqs

a disadvantage. This was one of the few topics in this question of the survey that the respondents largely agreed upon, and notably, two respondents strongly agreed with it. Only two individuals disagreed with this statement. Similarly, participants were largely in agreement that *creating layouts for diagrams in PlantUML* presents challenges. Although no participant strongly agreed with the statement, it was the most agreed-upon statement among the other disadvantages, with six individuals agreeing, one remaining neutral, and two persons disagreeing.

The *high degree of freedom*, which was seen as an advantage by some, was viewed as a disadvantage by others, suggesting that this aspect may be a double-edged sword, providing flexibility while leading to inconsistency and unpredictability. The participants held strong opinions, primarily disagreeing with the notion that there is a *lack of guidelines regarding what to write in the requirements*. However, it is noteworthy that two respondents strongly agreed with the statement.

Participants expressed neutrality considering *missing functionality* from the tool and that *improvements and bug fixes take a long time to implement*. Both statements had a large group being neutral and the rest being roughly split between agree and disagree. Another statement that was met with neutrality among the participants was the concerns with T-Reqs *not properly supporting multiple repositories*, which could cause issues with mapping and different ways of working (WoW in 5.9).

The requirement of *filling in tags and attributes for the requirements* did not seem to be universally recognized as a significant disadvantage, with responses showing disagreement with the statement. Two participants, however, do think this is a problem. The majority of participants also disagreed when posed with the statement that it's a disadvantage that T-Reqs is *reliant on standard IT technology like Git/Linux*, *integrated into the development environment*, *difficult to setup* and *difficult in regards to merge conflicts in Git*. The first two statements, each having four participants strongly disagreeing, four disagreeing, one person agreeing (*Respondent 2*), while the last remained neutral (*Respondent 1*). Regarding the distribution of difficulties with tool setup, a similar pattern was observed. In the case of difficulties with *merge conflicts in Git*, the majority disagreed, although it is worth noting that three participants reported experiencing this problem.

Upon evaluating the most significant perceived disadvantages, a clearer picture of the participants' concerns with T-Reqs emerges, the results being presented in Figure 5.10. The most frequently selected main disadvantages include the *lack of visualization for connections between requirements*, the difficulties *managing multiple repositories*, and *create layouts in PlantUML* with five selections each. The tool's *high degree of freedom* followed as second most frequently selected, with four participants expressing this as a main advantage. It is notable that the statements *lack of guidelines* and *resolving merge conflicts in Git* were largely disregarded as disadvantages (refer to 5.9), but both were selected by two participants as the main disadvantages. This indicates that few individuals struggle with merge conflicts and lack of guidelines, but those who do think it greatly affects the use of T-Reqs. Five of the proposed challenges were not picked at all, and one was added by a participant. This statement read,

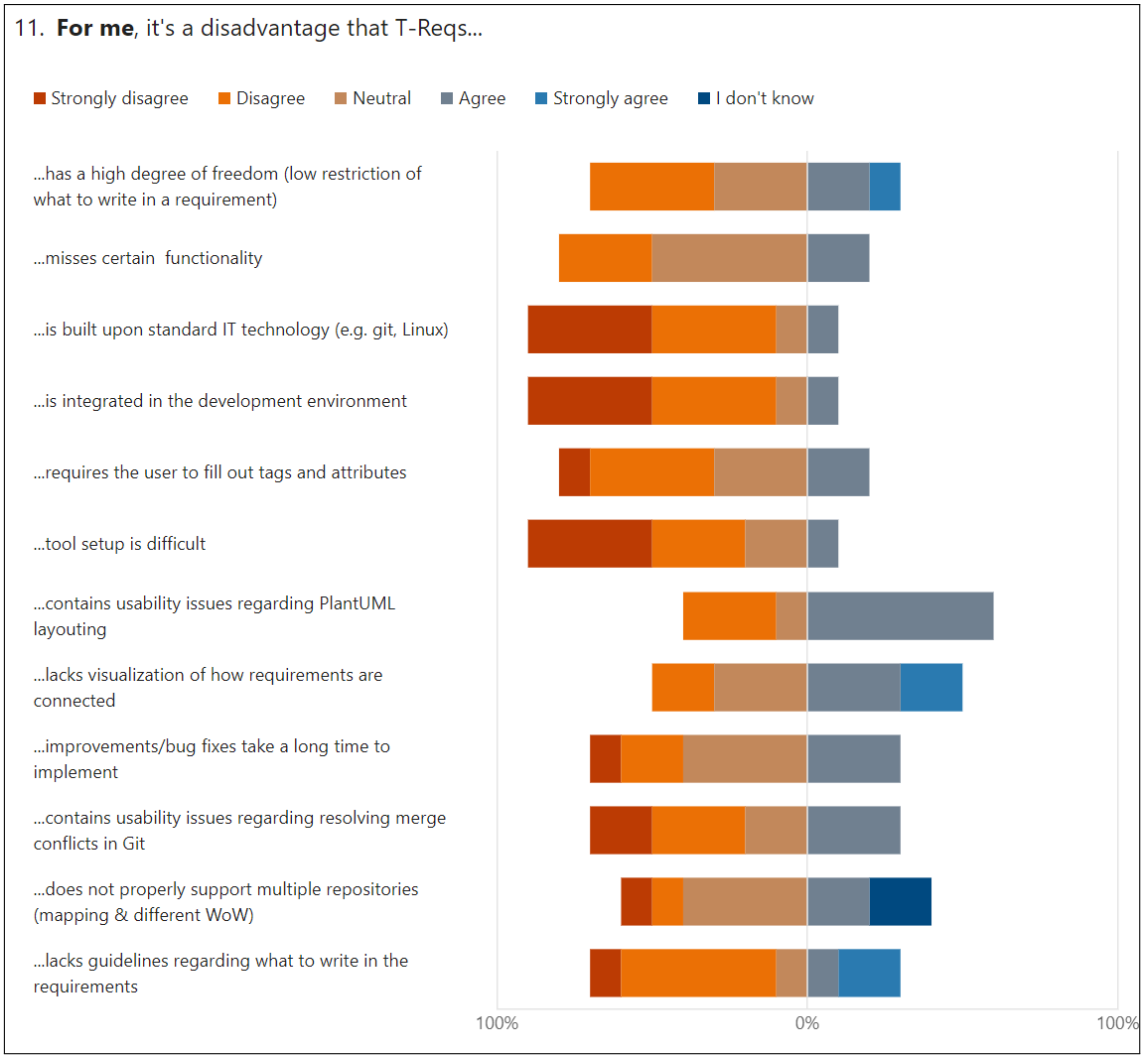

Figure 5.9: Diverging stacked bar chart displaying the responses to Question 11 in the survey related to disadvantages of T-Reqs

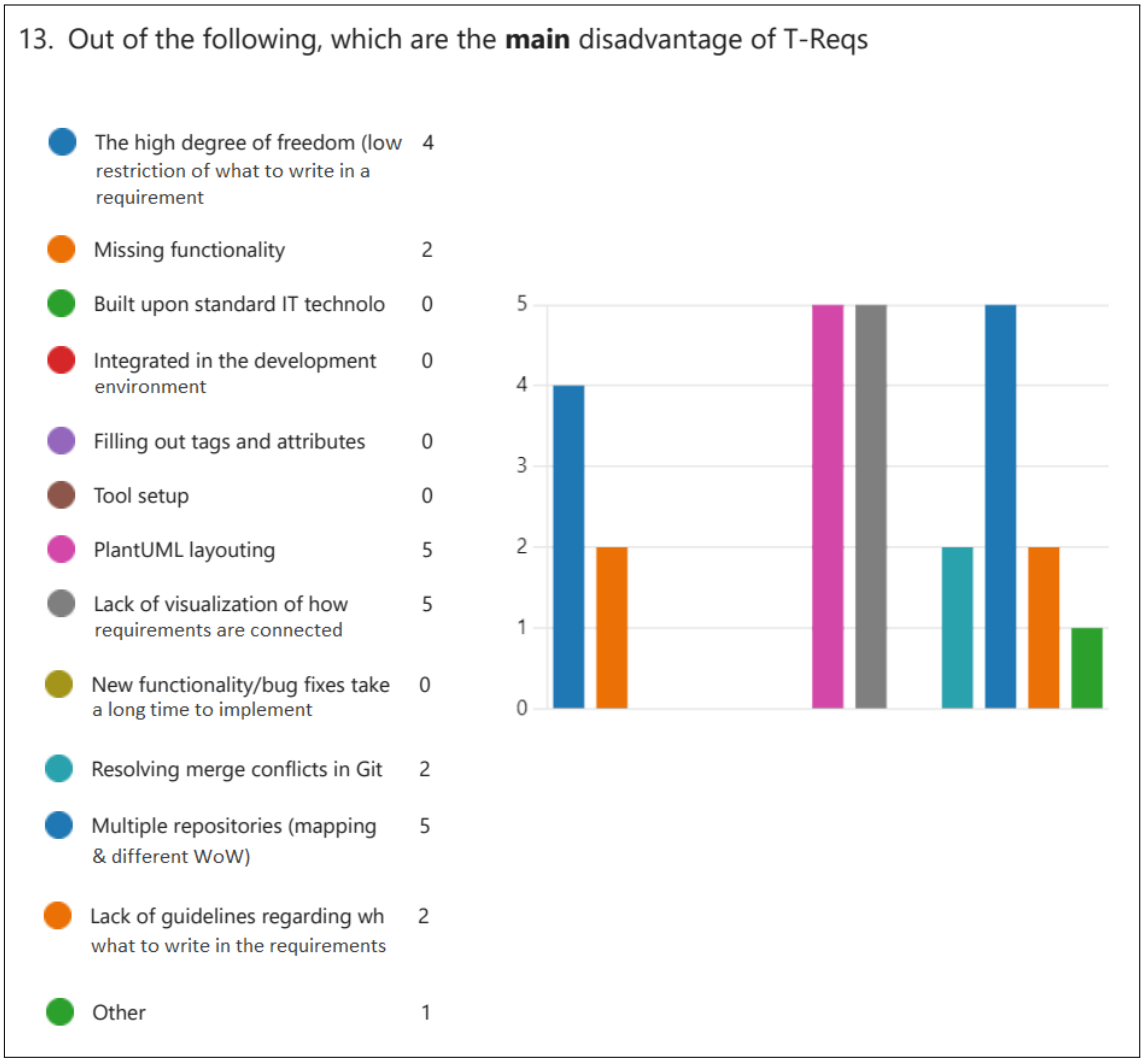

Figure 5.10: Multiple choice question from the survey for identifying the main disadvantages of T-Reqs

> "Lack of tracking to higher-level requirements." - Respondent 3

The participants were also asked to reflect on a set of contextual factors that may impact the successful adoption of T-Reqs, beyond the tool's intrinsic features. These factors include the collaboration between General System Managers (gen SMs in 5.11) and Team System Managers (team SMs in 5.11), the organizational culture at Ericsson, the company's continuous improvement of T-Reqs, and the assignment of responsibilities for handling requirements. The results of this survey section are presented in Figure 5.11. The majority of respondents agreed or strongly agreed that the collaboration between General System Managers and Team System Managers could indeed affect the quality of requirements, with only two individuals disagreeing, indicating a perception of the significant influence of teamwork on requirement engineering. Similarly, there was substantial agreement among the respondents that the organizational culture at Ericsson, particularly perceiving requirements as cumbersome, could potentially impact the timing of when requirements are written.

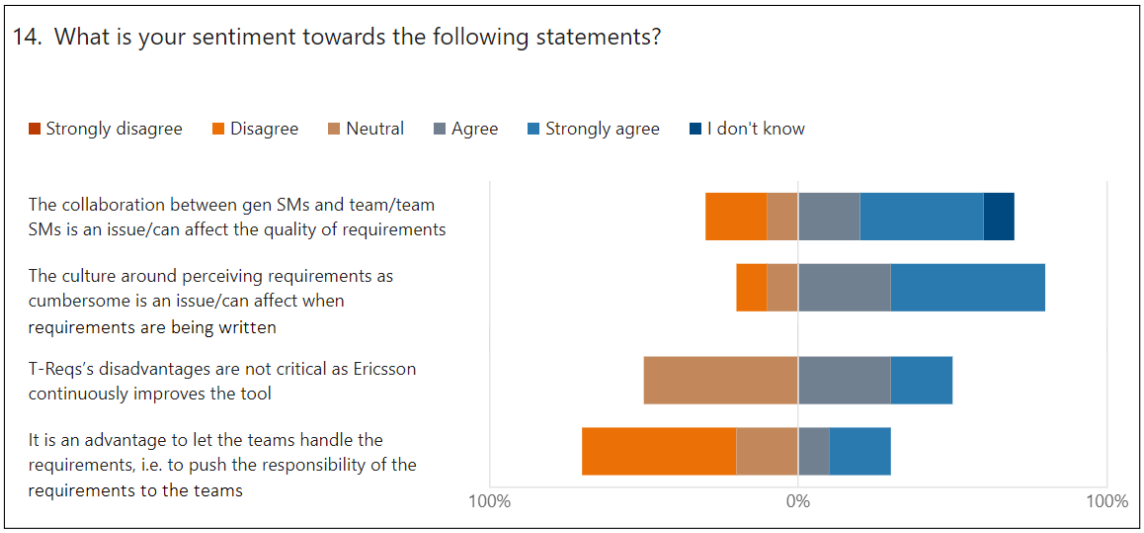

Figure 5.11: Diverging stacked bar chart displaying the responses to Question 14 in the survey for the contextual factors of T-Reqs's success.

Half of the participants took a neutral stance to the sentiment that the continuous improvement of T-Reqs at Ericsson mitigates some of the tool's shortcomings. The other half agreed or strongly agreed. This sentiment underlines the importance of ongoing tool refinement processes within the organization and aligns with the recognized advantage that T-Reqs, being developed in-house, allows for continuous improvements of the tool.

Responses were more diverse when asked whether it was advantageous to let the teams handle the requirements. This variation could reflect differing views on the distribution of responsibilities within the teams and the perceived benefits and challenges of this practice.

Figure 5.12 depicts the results obtained when participants were asked about factors influencing the quality of requirements. Respondents were divided between the requirement engineering tool with two votes, the way of working (WoW in Figure 5.12) with four votes, and the option of both with four votes. This demonstrates

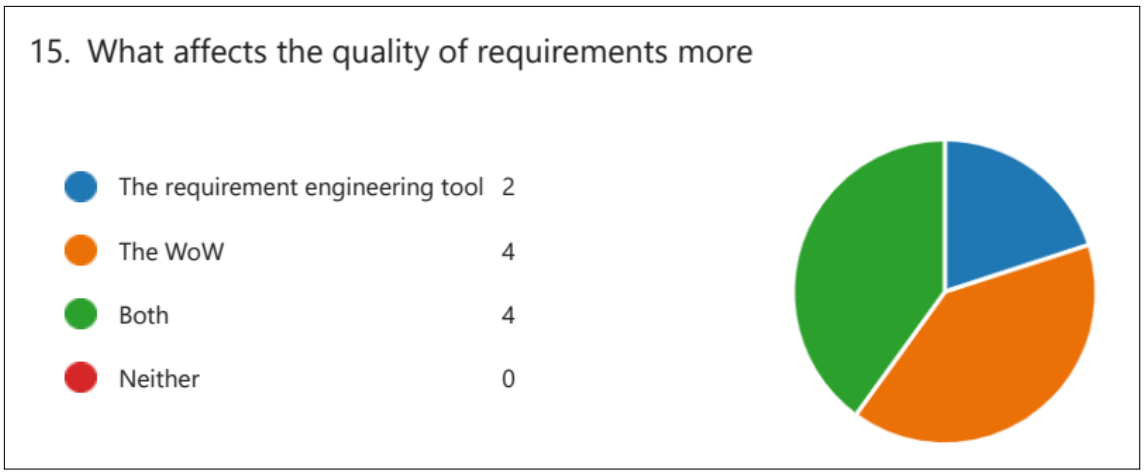

Figure 5.12: Pie chart illustrating the distribution of responses for Question 15 in the survey concerning what affects the quality of requirements

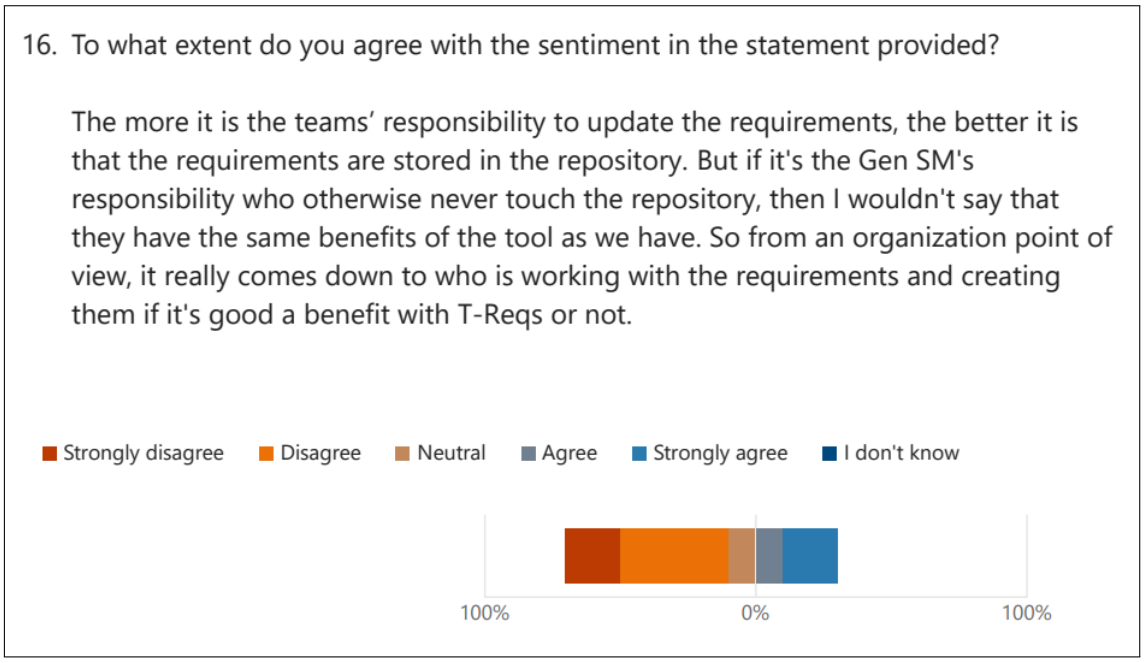

Figure 5.13: Diverging stacked bar chart displaying the responses to Question 16 in the survey regarding whether T-Reqs' success is connected to its architecture in relation to whose responsibility it is of handling the requirements

the recognition of multiple interdependent factors that contribute to the quality of requirements rather than attributing it solely to the tool or the method of work.

Lastly, respondents were asked to express their agreement or disagreement with a statement regarding the connection between the responsibility of updating requirements and the benefits of storing these requirements in a repository. The statement is based on a remark from one of the interview participants and can be viewed in Figure 5.13. Responses varied significantly, ranging from strongly disagree to strongly agree. This divergence might reflect the complexity of this issue, which appears to be contingent on various situational factors within the organization.

Overall, the survey results verified the positive perceptions of T-Reqs and its various advantages that were gathered during the interviews, while the disadvantages were largely disagreed upon. Further discussion and analysis of these results are presented in the Discussion, Chapter 6.

5.3 RQ3 (Characteristics)

What characterizes tools such as T-Reqs in contrast with more traditional requirements tools?

> The main characteristics of the tool noted by the interviewees and respondents of the survey were that the tool is connected to version control and integrated into each products' repository, provides ease of use as it uses technology familiar to developers, and resides in the development environment, and is text-file-based.

The difference between a feature and a characteristic is that a feature describes an important or interesting part of an aspect, while a characteristic describes a unique quality or trait of an aspect. Since some interviewees were unfamiliar with any requirements engineering tools except T-Reqs, they were instead asked to list what they considered to be the tool's key features. However, some of the characteristics the more experienced interviewees mentioned could not be considered unique to T-Reqs. Thus, they were classified as key features instead, i.e., as they were mentioned due to importance relating to the tool's functionality.

5.3.1 Characteristics Described by More Experienced Interviewees

The characteristics of T-Reqs, i.e., unique features, as identified by the interviewees who had previous experience with other requirements tools, are as follows:

Connected to Version Control: The majority of the interviewees mentioned the tool being integrated with version control as well as integrated into each products' repository as its characteristic and the main thing that sets it apart from previous tools. The absence of version control in the previous tools, which resulted in manual baselining, has thus been rectified and enables going back to previous versions of the product.

As most developers are familiar with using version control, it takes away their fear of losing work and also enables them to always work in the same environment, i.e., to not have to switch between the tools. Since it is connected to version control, it also facilitates work with other tools, such as Gerrit, for reviews and collaboration.

Ease of Use: As T-Reqs is built on standard IT technology that most developers are familiar with, i.e., Git, Linux, Markdown files, and UML diagrams, the tool can be characterized as easy to use. Especially since developers can stay in the same

environment while developing the product and writing requirements. Familiarizing yourself with the tool is then also made easier, as one interviewee stated,

> "...a fairly small set of rules to kind of understand how it is and then you can start writing your requirement documentation." - Interviewee 9

Additionally, since requirements are written as human-readable text, i.e., in Markdown files and using XML tags, it facilitates both writing and reading the requirements. This is contrary to the case in Rhapsody, as the diagrams could be outputted in documents as text which was, however, not human-readable. Furthermore, the storage of requirements in text files enables powerful modifications through a wide range of readily available tools.

Developed In-House: While not mentioned by all interviewees, most interviewees were positive towards the tool being developed in-house. By developing the tool in-house, there is more flexibility in adding and changing functionality than when companies use licensing tools, where requesting changes can be more challenging.

Bundle of Open-Source Tools: An interviewee described the tool as being a lightweight tool that includes a set of scripts and plugins,

> "...small concepts, small tools or utilities and you [..] knit it together..." - Interviewee 7

Additionally, T-Reqs was described as being a tight integration of a bundle of open-source tools, i.e., PlantUML, Git, and Gerrit, but being a blank canvas and thus offering a lot of freedom. This has, however, created some ambiguities as to where the boundaries of T-Reqs lie.

5.3.2 Key Features Described by More Experienced Interviewees

The key features of T-Reqs, as identified by the interviewees with prior experience with other requirements engineering tools, are as follows:

Different User Interfaces: T-Reqs offers different user interfaces or viewers, e.g., T-Reqs Web, T-Reqs Viewer and a plugin for the editors Eclipse and Visual Studio Code, mentioned in Section 3.2. The view most mentioned by the interviewees was the plugin for the editors. The plugin provides preview windows for the output documents, i.e., where the formatted result of the T-Reqs XML tags and the PlantUML in the Markdown files is displayed as HTML. The importance of T-Reqs Web was also mentioned, as was illustrated by one interviewee in the following remark,

> "...we've got the web-based interface which is very good, that's what I'm always using that's very easy to use [...]. Without that it would have been a problem, I think." - Interviewee 11

Displaying different views is, however, also a feature in traditional requirements engineering tools. As an example, RequisitePro offered both a table view and a document view. In addition, the tool also had a Word plugin.

Import & Export: One interviewee mentioned the importance of being able to import and export data from the tool, as the previous requirements data was imported from the previous systems that Ericsson had been using. Exporting the requirements documents as Excel and PDF documents is also possible. However, most requirements engineering tools offer these functionalities as well.

Traceability: Traces between requirements and test cases were mentioned as one of the more important features of the tool. This is, however, a main feature in the traditional requirements engineering tools and is thus not unique to T-Reqs.

Simple Automated Quality Checks: Automated quality checks were also mentioned as a valuable feature. These checks provide feedback and indicate, for example, whether the attributes associated with a requirement are not in order. The purpose of these checks is to stop incorrect formatting from being pushed to a particular branch. In T-Reqs, this is done by using a Gerrit plugin in Jenkins, which returns a score indicating whether the changes made have passed the quality check or not.

5.3.3 Key Features Described by Less Experienced Interviewees

Due to some of the interviewees not having prior experience with other requirements engineering tools than T-Reqs, they were asked to identify the tool's key features instead. They are as follows:

Traceability: The primary key feature mentioned by the interviewees was the trace links from requirements to test cases. One interviewee remarked that,

> "...it should serve the purpose of [the tool]." - Interviewee 12

The traceability of the tool facilitates the validation of the requirements, i.e., confirming that a requirement has been covered by a test case or not. This is a necessity as it can not be argued that a requirement has been implemented unless it can be traced to a test case.

Lightweight & Easy to Use: As a key feature of the tool, the less experienced interviewees mentioned that the tool was lightweight and easy to use but not always as easy to contribute to, i.e., due to inexperience with Git. One interviewee remarked that the more elaborate user interface of the licensing tools was not to their liking and that they liked simplicity instead, stating *"the simpler, the better" (Interviewee 10)*. A description of the tool given during the interviews was that it is *"developer friendly" (Interviewee 3)* as it is text-based, i.e., requirements are written inside of XML tags, and the file types are Markdown. The interface, the search functionality on the T-Reqs Web, and the rendering of Markdown documents to HTML previews were also mentioned as contributing features to the tool's usability.

Integration & Version Control: The integration of the tool into each products' repository and, thus, the whole development environment was another key feature men-

tioned. This facilitates the workflow due to Git version control being used to store the requirements and code together. It makes it also easier to access the requirements.

Modeling: The fact that PlantUML is supported by the tool enables developers to search and version control the UML diagrams constructed more easily. One participant stated that it was *"a necessary evil" (Interviewee 5)* and that perhaps a better proprietary tool existed, but it would then not be standard IT.

Developed In-House: The tool being developed in-house was also mentioned by the less experienced interviewees as an important feature of the tool, and thus it is included in this section. The interviewees stated that by developing the tool in-house, Ericsson could make the tool fit its environment better. Thus, offering functionality that is the most suitable for each product.

5.3.4 Results from the Survey

The final purpose of the survey was to validate the identified characteristics and key features that were mentioned by the participants during the interviews. This was done by providing all the features mentioned from the interviews and requesting that the respondent choose the ones that they considered to be the main characteristics and the key features of T-Reqs. Every respondent answered the questions posed.

Figure 5.14 displays how the respondents voted in terms of selecting the main characteristics of T-Reqs. The options provided were in accordance with the interview responses, that is, all features of the tool that the interviewees had identified as its characteristics, despite some not being unique to T-Reqs. The reason why we did not remove the features which were not unique was that we wanted to avoid interfering with the data collection, i.e., we wanted to stay true to the opinions of the participants. Thus, allowing us to confirm the findings from the interviews.

The main characteristics selected by the participants of the survey, each receiving six votes, were *Connected to Version Control* and *Integrated into Product's Repository*. The tool's characteristic of being *Text-File-Based* received five votes. Three characteristics received four votes, i.e., *Developed In-House*, *Ease of Use*, and *Simple Automated Quality Checks*. It should be noted that some type of quality management is present in other requirements engineering tools as well. However, the checks that are performed can be classified as being unique to T-Reqs as they validate the syntax of the requirement and whether certain attributes are present. One participant only selected the option "Other" due to a lack of experience with other requirements engineering tools.

In Figure 5.15, the responses from the participants can be seen indicating the most valuable key features of the tool. The options provided are again in accordance with the interview responses. Thus, this survey question aims to accurately confirm those findings.

The key features selected by all participants except one was that the tool is *Connected to Version Control*. The key feature with the second most votes, i.e., seven votes, was *Integrated into the Product's Repository*. Five features are tied

with four votes each, namely *Ease of Use*, *Text-File-Based*, *Traceability*, *Lightweight* and *Automated Test Reporting*.

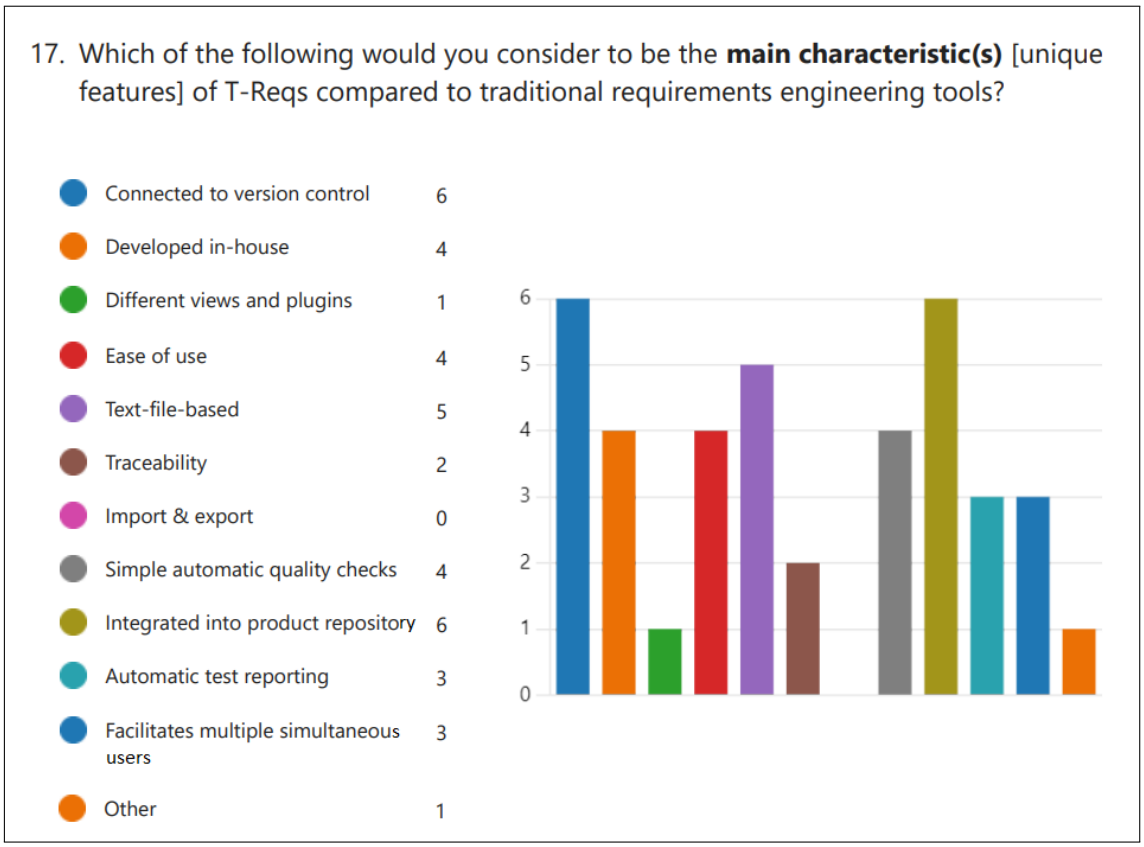

Figure 5.14: Multiple choice question from the survey for identifying the main characteristics of T-Reqs

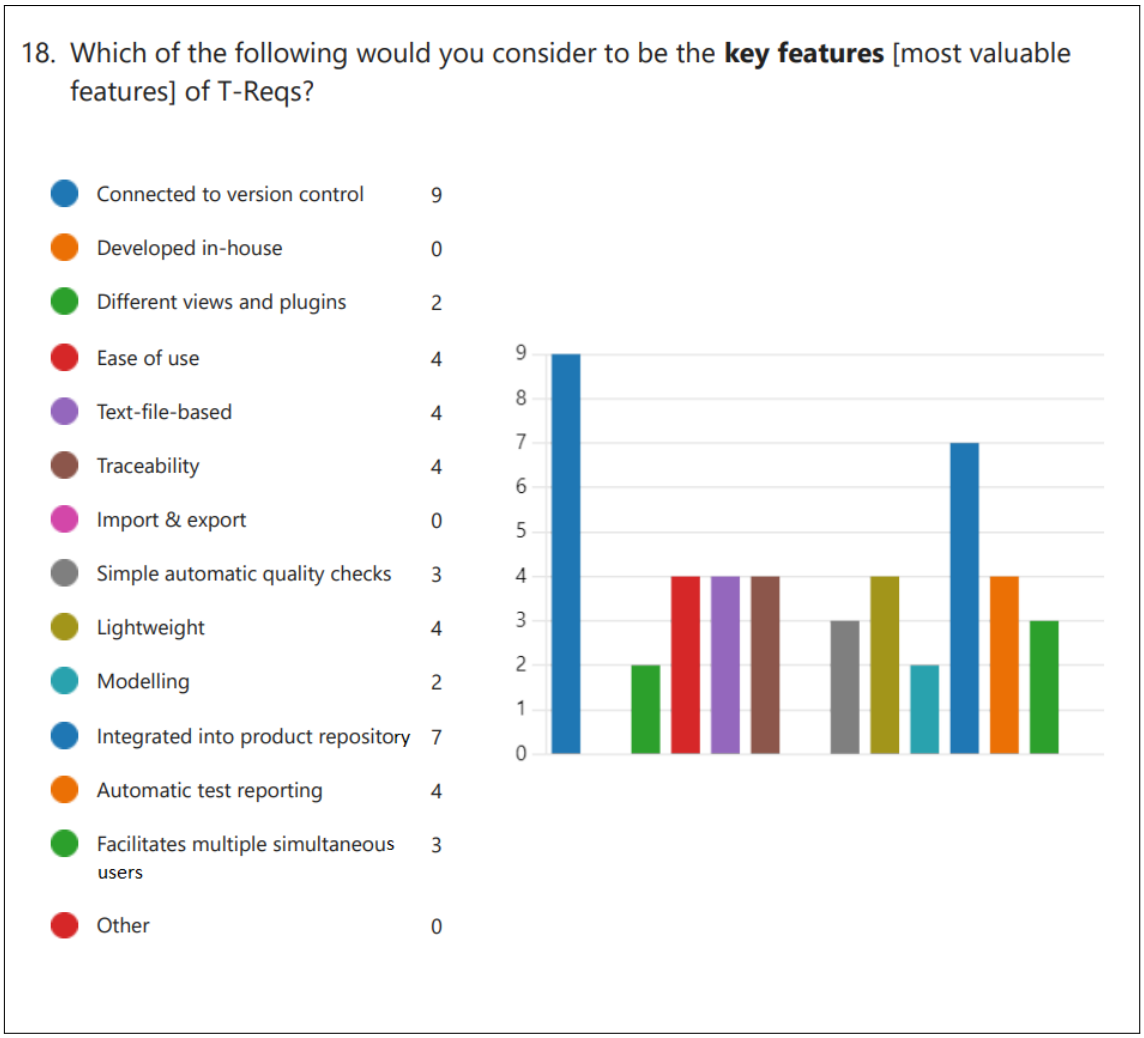

Figure 5.15: Multiple choice question for identifying the key features of T-Reqs

5.4 Results from the Workshop

The characteristic of T-Reqs that most resonated with the participants of the workshop was the strong connection that is formed between the requirements and the code and test cases by storing them all together. Thus, the requirements are connected to the development more closely and will, in turn, be an incentive to keep them updated.

In terms of context regarding who is expected to use the tool, the participants agreed it was not targeted towards personnel at a higher level in the development, i.e., managers, and rather developers working with the code. This corresponds with the goal of pushing the requirements' responsibility down to the development teams. This can, however, be problematic, as in Ericsson's case, when the General System Managers are still the artifacts responsible for the requirements documents. A solution brought up to better accommodate the employees working with requirements at a higher level and thus not necessarily possessing the technical knowledge to use the tool was to offer them a different Graphical User Interface (GUI). This would, in the case at Ericsson, be beneficial for the General System Managers if they wanted a quick way to, for example, create or update a certain requirement. The reason this has not been implemented at Ericsson is due to the edge cases that occur during merge conflicts which they have not found an easy way to solve. The participant from one of the automotive companies stated that they solved this conundrum using a different tool for high-level requirements than their more technical low-level requirements.

One of the participants from Ericsson came from an organization that had not yet implemented T-Reqs in their product domain. Instead, a centralized tool was being used where only one person was capable of managing the requirements for each release. Additionally, they mentioned that it was the custom of the team to do the requirements before the development.

The Manufacturing company develops software and hardware and thus has both software and system requirements. Currently, the software developers are using a requirements engineering tool deemed the industry standard, but they greatly dislike it and would instead want to use T-Reqs. T-Reqs is also particularly suitable to them since they work with Markdown and PlantUML. Therefore, as stated by the participant, if they were only developing software, they *"would jump straight into this" (Manufacturing Participant)*. The challenge, however, is that requirements are company-wide, encompassing both software and hardware development, which differ considerably. As such, T-Reqs as a tool would struggle to scale effectively across the entire organization. There is also resistance towards Git version control as the whole company's development does not rely on it. The key takeaway from this case context is that Ericsson managed to localize the requirements to each product which is not currently possible at this company.

6
Discussion

In this chapter, the interpretations of the results will be presented, as well as the implications for research and the industry. Furthermore, we aim to extend the applicability of our findings beyond the context of Ericsson by providing valuable advice on key considerations in relation to requirements engineering tools. To conclude this chapter, possible future work is discussed.

6.1 Interpretation of the Results

6.1.1 RQ1 (Expectations)

In order to give the context behind the goals and expectations of the tool, the considerations and decisions Ericsson made were illustrated as these decisions are made in order to meet a certain expectation of what the tool should do. The decisions were that the tool should be connected to version control to reduce manual labor that occurred in the previous tools, introduce a way to do modeling, have trace links between requirements and test cases, integrate the tool into each products' repository, offer interoperability with other systems, and allow multiple simultaneous users. The goals and expectations that Ericsson hoped to meet with these decisions were that the tool should be easy to use as well as improve their ways of working. More specifically, allow the company to scale-out, i.e., increase the number of development teams, have automated test reporting, facilitate collaboration, pertain the structure, and have the master branch release ready. Identifying these expectations made it possible to validate them from the interviewees' responses, and further verified in the follow-up survey. This study has shown that most of the initial goals and expectations of T-Reqs have been met. In terms of the goals that have not been fully met, the hindrance has not been the tool itself but rather the lack of experience with Git version control among some employees at Ericsson on the one hand and the long runtime of the test suite on the other hand.

During the initial stage of this case study, a literature review was conducted partly in order to analyze the context in which T-Reqs operates. The review yielded papers regarding challenges conducting requirements engineering in different agile development contexts, which were discussed in Section 2.2.1 and Section 2.3. In one of the aforementioned papers, Kasauli et al. [41] identified the requirements engineering challenges in large-scale agile system development. Additionally, solutions were suggested in order to mitigate these challenges. In accordance with these solutions, the identified requirements engineering challenges are related to the decisions

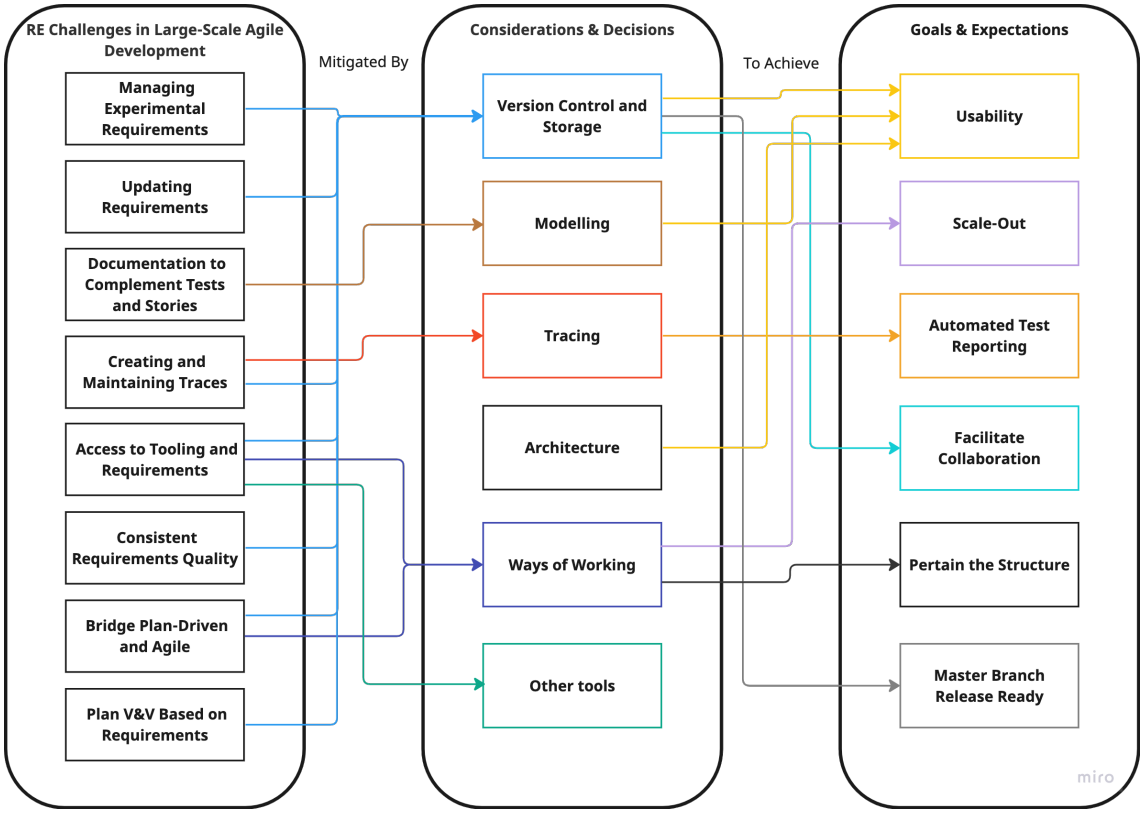

Figure 6.1: Connecting the challenges found in literature to the decisions and goals relating to T-Reqs

and goals from the results to RQ1 (Expectations) in Section 5.1 and displayed in Figure 6.1.

As is illustrated in the figure, all except one of the challenges are fully or in part mitigated by using version control and storing the requirements with the code and test cases. That is, the challenge of managing experimental requirements can be solved by using branches in Git. The challenges of updating requirements and creating and maintaining trace links are also made easy with Git, as additions and changes can be made compared to normal code, and each change is grouped together in a Git commit. Access to tooling and requirements is also facilitated, as standard IT technology, which is familiar to most developers, is employed, i.e., Git and Markdown files. Managing the requirements with the aid of version control also makes it possible to use tools such as Gerrit to review changes being made and add comments. This, in turn, enables to maintain consistent requirements quality as well as planning validation and verification based on the requirements. That is, since the requirements are stored with the test cases, and traces exist between them, they will remain consistent, and a change in either of the items will result in a need for a change in the other as well. The review process in Gerrit also makes it possible to bridge the gap between the more plan-driven approaches on the system level, i.e., where different feature studies are conducted, and the agile development on the team level. Since the General System Manager is the artifact responsible, they can be notified of any changes being made during the development process by the team and view changes clearly in Gerrit and give feedback to the team as well. In addition to version control assisting in creating and managing traces between requirements and the test cases, scripts are also in place to validate the links that have been added to the requirements documents, linking requirements to the test cases.

By moving the responsibility of managing the requirements down to the teams, the need for a tool to enable multiple simultaneous users is evident. This made it less feasible to use the previous traditional requirements tools charging fees per user. Expanding access to the tool and requirements also removes the bottleneck which was present when General System Managers were tasked with managing the requirements. Thus, this also mitigates the challenge of bridging the gap between plan-driven and agile, i.e., the process goes smoother by eliminating bottlenecks.

The challenge of providing enough documentation to offer a sufficient understanding of a complex system can be mitigated by allowing developers to create models of expected system behavior. In the case of T-Reqs, the models are created with PlantUML, which can be integrated into the requirements documents.

The decisions made can also be related to the goals that were set for the new tool, i.e., decisions are made with what Ericsson wanted to achieve in mind. The decision of wanting to have version control integrated into the tool and thus deciding to store the requirements with the code and test cases in Markdown files was to facilitate ease of use. That is, using standard IT technology and allowing the developers to manage the requirements in the same environment as the development since the tool is integrated into each product's repository. It also facilitates collaboration among the General and Team System Managers as Gerrit reviews can be utilized. The decision behind where to store the requirements was also intended to enable

the code and requirements to always be in sync and thus have the master branch release ready at all times. The usability is increased further by allowing the creation of UML diagrams to illustrate the intended behavior of a system. The decision to have traces between requirements and test cases made it possible to achieve the goal of offering automated test reporting in the tool with the help of scripts. With the decision to move towards more agile ways of working and wanting to integrate requirements management more with the development processes, Ericsson was able to scale-out the company by introducing more development teams. Additionally, the decisions made regarding the new ways of working were constructed with the goal of wanting to pertain the structure of requirements processes in mind. That is, requirements documentation and management should still be prioritized and not lost despite moving towards a more agile development environment.

As the goals could be mapped onto the illustrated challenges through the consideration and decisions made in regard to the implementation of the tool, it further strengthens the tool's value in this development environment. That is, by achieving the initial goals and expectations, the tool is also mitigating some of the challenges faced in large-scale agile system development. This study also, in turn, confirmed that the suggested solutions posed by Kasauli et al. have successfully mitigated, for the most part, these challenges in an industry context.

As illustrated in the case study performed by Bjarnason et al. [15], integrating requirements engineering processes into the development process mitigates the challenge of communication issues occurring. That is, they note that the experience is that when different roles within a company readily discuss requirements, a common understanding is reached. The results of this study further support this claim. That is, according to the results from the interviews as well as the survey, T-Reqs has facilitated collaboration between the General System Managers and the cross-functional teams. However, in order to have accessible communication, all parties must be willing to partake in the discussion and usage of the tool. Further, the right people have to be notified at the right time.

In the empirical study by Ramesh et al. [16], the importance of managing requirements change was emphasized, i.e., requirements must be susceptible to change during the development process. As agile development is, at its core, iterative, the notion of storing the requirements with the code and test cases in a version-controlled environment thus ensures that change is expected to all artifacts in question. That is, if the behavior of the code must be changed so must the test cases validating the code and, thus, the requirements linked to those test cases. Therefore, every artifact will be in sync at all times during development, and this feature is what T-Reqs offers its users. However, it should be noted that this hinges on the requirements being, to some extent, written before the development starts. In the case at Ericsson, interviewees noted that this was not always the case and mentioned occasions where requirements were not written until towards the end of the development. Thus, it is essential to establish guidelines for how the tool should be used in addition to the functionalities that it offers.

From these results, it can be deduced that T-Reqs and tools such as T-Reqs would offer value and facilitate managing requirements in large-scale agile system development. However, each company needs to analyze its context and goals re-

garding requirements engineering practices and select a tool based on those aspects. Further considerations are given in Section 6.2 below.

6.1.2 RQ2 (Advantages and Disadvantages)

As we delved into the real-world application of T-Reqs, a compelling narrative emerged from the interviews, survey, and workshop results. The users primarily expressed satisfaction with T-Reqs and its overall functioning. It is worth noting, however, that the feedback recurrently highlighted the room for improvement not in the tool per se but in the surrounding processes.

T-Reqs, being integrated into the development environment and employing technology familiar to the users, emerged notably developer-friendly. This advantage resonates with the statements from the previous studies on T-Reqs, highlighting the familiarity aspects it provides for development teams [7], endorsing the inherent value of leveraging pre-existing IT technologies and fostering integration within the development environment. The same study noted certain concerns with model handling in T-Reqs, inviting future research. This matter is elaborated upon by Liebel et al. [9] in their case study, wherein they explore effective approaches for integrating modeling within an agile environment. Our study adds further dimensions to this narrative. We illuminate the complexities involved in model handling, with the application of PlantUML emerging as a practical, albeit imperfect, solution. What has been identified is that it is easy to manage models in T-Reqs due to their text-based nature (PlantUML), thereby enabling version control. However, other file types, such as images, are not to be appended beyond PlantUML diagrams as these cannot be version controlled to the same extent. One area where PlantUML falls somewhat short concerns user aspects, particularly in terms of creating layouts, which can be challenging to achieve as desired. This trade-off, integrating the modeling in the software development workflows, contra difficulties controlling the diagram layouts, was also identified in the aforementioned paper by Liebel et al. [9]. Our respondents indicated that models were not of primary importance, they were considered beneficial. The advantages of PlantUML appear to outweigh its disadvantages, and none of the participants could suggest an alternative to the current modeling solution.

The users' desire for refinements was mainly directed at optimizing these PlantUML layouts but also enhancing the visualization of requirement connections. The tool's collaborative features, such as multi-user access and review processes, were met with considerable appreciation. The same goes for the tracing abilities of T-Reqs.

Despite the generally positive feedback, several organizational areas of improvement were identified. There is a pressing need to establish clear guidelines, especially for non-straightforward issues, in order to maintain the quality and consistency of requirements. An example of such an issue is the level of detail in requirements, which recurred as a discussion topic. In Ericsson's context, this pertains to the level of detail in the artifacts transferred from the General System Manager to the development team. Opinions varied regarding the optimal combination of artifacts to be handed over to the development team, which might include initial studies

(presented in PowerPoints), structured documents for requirements, or high-level requirements in T-Reqs. Some participants emphasized the need to limit high-level requirements to avoid overwhelming the General System Managers with work and to prevent an overpopulation of high-level requirements in the documents. Others, however, preferred to ease the process of writing requirements for the team. The level of detail in the requirement writing process by the team also necessitates explicit guidelines, as it has been mentioned that requirements have been written with too much detail. Our study exposed a gap in clarity on the responsible entities, appropriate timings, and content of requirements writing.

Furthermore, an important aspect for discussion is the potential challenge posed by the lack of dedicated time for working on requirements, combined with the prevailing perception that requirements are often redundant. This perception may result in lower priority being assigned to requirements, causing their creation and maintenance to be delayed and pushed further into the development process. In the case of Ericsson, the responsibility for requirements was delegated to the team due to the overload on General System Managers. However, the Team System Manager, who holds the requirements perspective within the team, often perceives requirements as a secondary task in addition to their regular development responsibilities. Reports indicate that requirements are frequently written towards the end of the development cycle, with some participants disclosing that they have yet to experience a normal development cycle. Despite the majority of participants expressing a preference for early requirement definition, it becomes evident that while requirements are desired, insufficient time is allocated for their realization. In this context, the widespread access to T-Reqs could inadvertently amplify the issue, given that the responsibility for managing requirements could easily be passed on under the assumption that someone else will handle it.

Ericsson employees have reported numerous occasions where requirements management was deprioritized for various reasons and consequently created requirements debt. This corresponds to the findings of Heikkilä et al. [10] of an increased amount of technical debt after an agile transformation. Interestingly, this occurred both during the period when Ericsson was using Rhapsody and in recent years after T-Reqs was adopted. In these described situations, it was mostly Ericsson's processes that affected the deterioration of requirements management. This deterioration in the first case was due to the discontinuation of Rhapsody when the requirements became overly detailed, coupled with low pressure from management to update them. In the second case, T-Reqs was used inconsistently during a certain period when a new product needed to be developed quickly. The company entered a fast-paced ramp-up phase with many new teams, which led to a low priority for requirements management. This, unfortunately, led to a malfunction in the product as no requirements existed for the implemented feature. These examples, which exhibit two distinctly different requirements management tools, underscore the importance of a structured requirements management process with clear guidelines. This is not to diminish the tool's relevance in the trajectory of the requirements engineering process, as, for example, part of the downfall in using Rhapsody was due to the time-consuming nature of updating requirements and the inherent nature of modeling tools not scaling well with complexity. However, it does highlight the notion that

any tool can be mishandled. Rhapsody's case also illustrates that the requirements engineering tool's usability aspects are critical for a consistent and sustainable requirements engineering process. Thereto the time to complete a task in the tool and the usefulness of the finished requirement.

To summarize, explicitly stating the *who*, *what*, and *when* aspects of writing and updating requirements within guidelines and standard procedures is crucial for establishing a sustainable requirements engineering process, utilizing the advantages of using a requirements engineering tool in the first place. This sentiment is further supported by Muhammad et al. [27] in their design science research, where guidelines for defining a requirements strategy in an agile environment were established. The paper stated that "...a requirements strategy should describe how requirements are structured, how work is organized, and how [requirements engineering] is integrated in the agile work and feature flow." (p. 87).

6.1.3 RQ3 (Characteristics)

The main characteristics identified after the validations of the results in the survey were that the tool is connected to version control, integrated into each products' repository, and text-file-based. In addition to those characteristics, the tool being easy to use, developed in-house, and having simple automated quality checks were also highly ranked.

It can be debated whether the tool's ability to perform simple automated quality checks is a characteristic or a key feature. It is certain that the checks that are performed can be classified as being unique to T-Reqs as they validate the syntax of the requirement and whether certain attributes are present. However, different quality management exists in other requirements engineering tools as well. Additionally, the tool being developed in-house is specific to T-Reqs and, thus, not a characteristic of the other tools with which T-Reqs can be grouped. The remaining identified characteristics, i.e., the ones that received the most and second most votes, can be associated with T-Reqs and tools similar to T-Reqs. That is, Doorstop is described as a "requirements management tool that facilitates the storage of textual requirements alongside source code in version control" [5]. An alternative to T-Reqs and Doorstop is rmtoo, which is described as a "free and open source requirements management tool" [44], which uses a command line interface and can be integrated with version control systems. It is further stated that its benefit is that it resides in the development environment. Thus, reaffirming that the identified characteristics encompass the other tools in the class of tools which T-Reqs is grouped into.

As discussed regarding the topic of agile requirements engineering in Section 2.3, it can be challenging to discuss a topic without having established a universal definition. Thus, to facilitate discussion regarding these emerging requirements engineering tools, a formal definition is suggested with the identified characteristics and descriptions of the tools above in mind.

Definition 1. ***Text-based requirements management tool*** *is a distributed, text-file-based requirements engineering tool for documenting and managing requirements which are stored next to the code and test cases in order to rely on version control for baselining and remaining in the same environment as the development process.*

The categorization offered in the paper on Doorstop [4] could also be employed, i.e., to classify these tools as decentralized as opposed to the traditional requirements engineering tools being centralized. However, as these tools offer multiple simultaneous users who can work independently, they can also be classified as distributed.

6.2 Advice for a Broader Context

When it comes to implementing a requirements engineering tool in an organization, several critical considerations need to be addressed. This advice section aims to provide insights that go beyond Ericsson's context based on the findings of this study, especially the results of a cross-industry workshop. It is crucial to remember that each company's situation and needs are unique, and advice should be tailored accordingly.

6.2.1 Understand the Organizational Context

The successful adoption of a requirements engineering tool largely depends on its compatibility with the organizational context [28, 31, 32, 34]. Various factors should be carefully considered to ensure an optimal fit. These factors include the size of the organization, the complexity of projects, the nature of requirements, and the level of required collaboration can significantly influence the suitability of a tool. In some cases, it may be beneficial to decentralize the requirements process and localize it to each product, as Ericsson did, to cater to product-specific needs. However, this might not be possible or desirable for all organizations.

6.2.2 Consider the Users

The potential user of the tool is a crucial consideration. That is, understanding the technical proficiency of the users, their familiarity with the tool or technology used in the tool, and their general work environment is critical. A tool should be selected that is convenient and intuitive for the users and fits well within their existing work ecosystem. For instance, if the tool is geared towards developers, like T-Reqs, it might alienate higher-level personnel. Offering different interfaces to accommodate different user types, such as a graphical user interface for non-technical staff, might be a good solution.

6.2.3 Focus on Collaboration

In the era of agile and distributed development, collaboration plays a critical role in requirements engineering. A tool that enables seamless collaboration, simultaneous multi-user access, and an integrated review process can significantly enhance the efficiency and effectiveness of requirements engineering.

6.2.4 Cost-Benefit Analysis

A comprehensive cost-benefit analysis is essential for making an informed decision about the adoption of a requirements engineering tool. This analysis should consider both the licensing costs of commercial tools and the costs associated with developing and maintaining a bespoke tool. For smaller organizations or less complex projects, the cost of dedicated tool development personnel may outweigh the benefits. However, for larger organizations or more complex projects that have experience in developing software tools, the flexibility and adaptability of a custom-developed tool may justify the costs and, in the end, possibly be more cost-effective than a licensed tool. There is also the option to choose one of the open-source tools similar to T-Reqs on the market, but modifications might be needed.

6.2.5 The Role of Guidelines

Adopting a flexible and unstructured tool offers numerous benefits, such as avoiding time-consuming and frustrating processes and promoting agile ideologies by instilling user empowerment. However, without clear guidelines, it can also lead to inconsistencies and quality issues. Thus, when adopting such a tool, organizations should ensure clear and exhaustive guidelines for requirement writing and management.

6.2.6 Final Advice

It is important to remember that the success of a tool does not solely depend on its features and capabilities. A supportive organizational culture, clear guidelines, and effective collaboration are equally important for successful requirements engineering. These human and organizational aspects should be given equal attention to the technical aspects of the tool. Thus, the choice of a requirements engineering tool should not only align with the organizational needs but also contribute to fostering a positive and productive work environment.

6.3 Implications for Research and the Industry

The research conducted situates T-Reqs within a broader, more detailed context. It verifies previous findings and complements them with a user-centric analysis, enriching the discussion on tooling in agile development and providing an extensive view of a tool and its users in a specific context. These real, unabstracted details allow readers to extrapolate lessons applicable to their respective contexts, aside from the insights and advice that are given.

6.3.1 Research

This in-depth case study contributes to the research community in the area of requirements engineering in large-scale agile development. In addition, the study contributes to the current research on text-based requirements engineering tools, which is currently scarce compared to traditional requirements engineering tools. Moreover, it provides an exploration of an alternative to traditional requirements engineering tools, specifically in large-scale agile system development. It builds upon the previous research of challenges regarding both the development context and agile requirements engineering by confirming the possible mitigation of them by using tools such as T-Reqs. As it can be challenging to discuss a certain term or topic without a universal and formal definition, e.g., as has been mentioned regarding agile requirements engineering, a formal definition of tools such as T-Reqs will facilitate the discussion regarding these requirements engineering tools in a scientific manner.

6.3.2 Industry

The results of this case study and the advice given for a broader context (in Section 6.2) provide practitioners an insight into possible alternatives to traditional requirements engineering tools. That is, facilitating the selection process for companies by illustrating in which context tools such as T-Reqs can be applicable and thus determining whether they are suitable in their company context.

When a company is in the process of selecting a tool, it's beneficial to understand the context in detail, especially if the context is similar to that of the company, but not necessarily. Therefore, abstracting irrelevant details and extracting lessons that fit the reader's context is crucial to understanding problems and their solutions, in this case, selecting a requirements engineering tool.

Given the identified lack of research on text-based requirements engineering tools, this study could be of great assistance even for companies not intending to adopt T-Reqs, simply due to the in-depth insights it can provide. For instance, observing what users of a requirements management tool focus on when it comes to various aspects of this type of tool, such as which functions are appreciated or less appreciated and what is considered user-friendly, can be valuable. It also makes it easier for people in the industry to reflect on requirements engineering tool structures and design and how these appear to users, both through our analysis of T-Reqs and the other tools mentioned (e.g., Rational RequisitePro and Rhapsody). The criteria list presented may not be extensive, but the truncated approach used provides a good and objective overview of several different tools in use today (T-Reqs, Doorstop, Jama, DOORS). We have also unearthed more general insights through this study, for instance, the importance of processes and considerations in conjunction with the tool. As the study has been conducted in a real-world environment by an impartial observer, the authors intend for the results to be of great value in the area of requirements engineering and requirements engineering tooling. Given that much of the information about requirements management tools on the market is either biased, i.e., advertising from tool vendors, or presented in impartial criteria lists that lack depth from a real-world environment.

In this report, the use of T-Reqs in a large-scale agile environment has been accounted for. Furthermore, T-Reqs has been defined as a text-based requirements management tool, coupled with a formal definition. Providing a definition for these tools (e.g., T-Reqs, Doorstop) makes them more visible as it brings to light their differences from other tools. Merely having a way to talk about them can impact the perception of these tools in the industry. In addition, we make them visible in a very straightforward way by showing that they exist to practitioners who are not familiar with these tools. Both aspects contribute to people making a more informed decision about what kind of tool is needed in their context.

For Ericsson, this study provides additional perspectives. The main advantages include a verification that it was worthwhile for Ericsson to develop T-Reqs and to continue supporting the tool, and that developing their own tools is an ongoing option for the future. In addition, identifying shortcomings and improvements in T-Reqs can contribute to a better requirements management process at Ericsson.

6.4 Future Work

As this research has only focused on investigating one requirements engineering tool at one company, the results illustrated could be used to compare how requirements engineering tools are used at other companies in the industry today and additionally try to draw conclusions of what tools are recommended for certain contexts. Further, it could be analyzed how widespread these text-based requirements management tools are or if traditional requirements engineering tools are still the industry standard. Expanding on our advice, guidelines for how companies should introduce a tool such as T-Reqs to their development and requirements engineering process could also be beneficial.

Maintaining the consistent quality of requirements in different repositories is a challenge, as noted in the interviews and the survey results. This also relates to the challenge of mapping requirements between different repositories, more specifically between high-level requirements and low-level requirements. How exactly this mapping is best implemented could be addressed in future research.

7
Conclusion

In this study, we have endeavored to evaluate T-Reqs, a modern requirements engineering tool, in its real-world implementation context. We have sought to answer our research questions related to how well the tool's performance matched the expectations before its implementation, its advantages and disadvantages observed in practice, and the characteristics that differentiate it from traditional requirements engineering tools.

The consideration and decisions before the tool's implementation were identified, as well as initial goals and expectations towards the tool. This was done in order to accurately answer the first research question regarding whether these expectations have been met. The results illustrated that all goals were either fully or partially achieved. Those expectations that were only partially met related to usability, as some users lacked experience with using Git version control and the expectation of wanting the master branch always to be release ready, which is currently not achievable due to an expensive test suite.

The investigation into the second research question yielded favorable responses towards T-Reqs from its users, highlighting its integration with the development environment, developer-friendly nature, and the tool's collaborative features as major advantages. The study also illuminated certain complexities and areas of improvement, including enhancing requirement connections' visualization, optimizing PlantUML layouts, and establishing clear guidelines for writing and updating requirements.

In addition to identifying the advantages and disadvantages of the tool, the characteristics of T-Reqs and tools similar to it were identified. This was in order to illustrate what makes these tools unique and sets them apart from traditional requirements engineering tools. These characteristics included the tools' strong connection to version control and closeness to code and test cases as well as being text-file based. A definition of this group of tools was also proposed, as well as a name to refer to them by, i.e., text-based requirements management tools.

This case study not only provides valuable insights for research and the industry but also outlines potential future work. Future research could focus on comparing how requirements engineering tools are used across different companies or investigating the adoption rate of these new tools versus traditional ones.

Our study concludes that the success of requirements engineering tools such as T-Reqs is heavily dependent on numerous factors, including the tool's features and capabilities, the organizational context, and the guidelines set for its use. Thus, organizations planning to adopt such tools should thoroughly evaluate their needs and context to ensure an optimal fit.

In conclusion, T-Reqs has demonstrated a successful implementation within its specific context, providing many advantages over traditional requirements engineering tools. Furthermore, it is a prime example of how modern requirements engineering tools can facilitate the development process in large-scale agile environments once their implementation is carefully tailored to the unique needs and context of the organization.

Bibliography

[1] Leffingwell, D. (2010). Agile software requirements: lean requirements practices for teams, programs, and the enterprise. Addison-Wesley Professional.

[2] Meyer, B. (2014). Agile! The Good, the Hype and the Ugly. Springer.

[3] Heikkila, V. T., Damian, D., Lassenius, C., & Paasivaara, M. (2015). A Mapping Study on Requirements Engineering in Agile Software Development. In 41st Euromicro Conference on Software Engineering and Advanced Applications, p. 199–207. doi: 10.1109/SEAA.2015.70

[4] Browning, J. & Adams, R. (2014). Doorstop: Text-Based Requirements Management Using Version Control. Journal of Software Engineering and Applications, 7, p. 187-194. doi: 10.4236/jsea.2014.73020.

[5] Doorstop. https://doorstop.readthedocs.io/en/latest/. Last visited 28th May 2023.

[6] https://gitlab.com/treqs-on-git/treqs-ng. Last visited 28th May 2023.

[7] Knauss, E., Liebel, G., Horkoff, J., Wohlrab, R., Kasauli, R., Lange, F. & Gildert, P. (2018). T-Reqs: Tool Support for Managing Requirements in Large-Scale Agile System Development. In Proceedings of 26th IEEE International Requirements Engineering Conference, Demo Track, Banff, Alberta, Canada. doi: 10.48550/arXiv.1805.02769.

[8] Gren, L. & Lenberg, P. (2020). Agility is responsiveness to change: An essential definition. In Proceedings of the Evaluation and Assessment in Software Engineering, p. 348–353.

[9] Liebel, G. & Knauss, E. (2023). Aspects of Modelling Requirements in Very-Large Agile Systems Engineering. Journal of Systems and Software, vol. 199, p. 111628, ISSN 0164-1212. doi: 10.1016/j.jss.2023.111628.

[10] Heikkila, V. T, Paasivaara, M., Lasssenius, C., Damian, D. & Engblom, C. (2017). Managing the requirements flow from strategy to release in large-scale agile development: a case study at Ericsson. Empirical Software Engineering, 22(6), p. 2892–2936. doi: 10.1007/s10664-016-9491-z.

[11] Runeson, P. & Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering, 14(2), p. 131-164. doi: 10.1007/s10664-008-9102-8.

[12] Creswell, J. J. & Creswell, J. D. (2018). Research Design. Qualitative, Quantitative and Mixed Methods Approaches. Fifth Edition. SAGE Publications, Inc.

[13] Saldaña, J. (2009). The Coding Manual for Qualitative Researchers. SAGE Publications, Inc.

[14] Inayat, I., Salim, S. S., Marczak, S., Daneva, M. & Shamshirband, S. (2015). A systematic literature review on agile requirements engineering practices and challenges. Computers in Human Behavior, Volume 51, Part B, p. 915-929, ISSN 0747-5632, https://doi.org/10.1016/j.chb.2014.10.046.

[15] Bjarnason, E., Wnuk, K., & Regnell, B. (2011). A Case Study on Benefits and Side-Effects of Agile Practices in Large-Scale Requirements Engineering. Agile RE 2011. Lancaster, UK: ACM.

[16] Ramesh, B., Baskerville, R., Cao, L. (2010). Agile requirements engineering practices and challenges: an empirical study. Information Systems Journal, 20(5), p. 449–480. doi: 10.1111/j.1365-2575.2007.00259.x.

[17] Measey, P. (2015). Agile Foundations : Principles, practices and frameworks, edited by Peter Measey. BCS Learning & Development Limited. ProQuest Ebook Central, https://ebookcentral.proquest.com/lib/chalmers/detail.action?docID=1759633.

[18] Flewelling, P. (2018). The Agile Developer's Handbook : Get More Value from Your Software Development: Get the Best Out of the Agile Methodology. Packt Publishing, Limited. ProQuest Ebook Central, https://ebookcentral.proquest.com/lib/chalmers/detail.action?docID=5314635.

[19] Hull, E., Jackson, K., & Dick, J. (2011). Requirements Engineering. Springer London.

[20] Clancy, T. (1995). The standish group report. Chaos report.

[21] Konrad, S. & Gall, M. (2008). Requirements Engineering in the Development of Large-Scale Systems. 16th IEEE International Requirements Engineering Conference, p. 217-222. doi: 10.1109/RE.2008.31.

[22] CPRE Foundation Level - Handbook. Retrieved from https://www.ireb.org/en/downloads/#cpre-foundation-level-handbook.

[23] International Organization for Standardization. (2018). Systems and software engineering — Life cycle processes — Requirements engineering (ISO/IEC/IEEE standard No. 29148). Retrieved from https://www.iso.org/standard/72089.html.

[24] International Organization for Standardization. (2009). Information technology — Systems and software engineering — Guide for requirements engineering tool capabilities (ISO/IEC TR standard No. 24766). Retrieved from https://www.iso.org/standard/51041.html.

[25] Glinz, M. (2022). A Glossary of Requirements Engineering Terminology. Version 2.0.1. Retrieved from https://www.ireb.org/downloads/#cpre-glossary.

[26] GPT-4 is OpenAI's most advanced system, producing safer and more useful responses. https://openai.com/product/gpt-4. Last visited 27th May 2023.

[27] Muhammad, A.P., Knauss, E., Batsaikhan, O., Haskouri, N.E., Lin, YC., Knauss, A. (2022). Defining Requirements Strategies in Agile: A Design Science Research Study. In: Taibi, D., Kuhrmann, M., Mikkonen, T., Klünder, J., Abrahamsson, P. (eds) Product-Focused Software Process Improvement. PROFES 2022. Lecture Notes in Computer Science, vol 13709. Springer, Cham. doi: 10.1007/978-3-031-21388-5_6

[28] Seilevel. (2016). Requirements Management Tool Evaluation Report. https://assets.cdnma.com/13314/assets/Website%20Downloads/2016-Seilevel-RequirementsTool-Evauation-Report-FINAL.pdf

[29] Fuggetta, A. (1993). A classification of CASE technology. In Computer, 26(12), p. 25-38. doi: 10.1109/2.247645.

[30] Santillán, M. F. (2015). Criteria for the evaluation of requirements management tools supporting distributed software product line engineering and management. https://jyx.jyu.fi/bitstream/handle/123456789/46250/1/URN%3ANBN%3Afi%3Ajyu-201506092243.pdf

[31] Heindl, M., Reinisch, F., Biffl, S. & Egyed, A. (2006). "Value-Based Selection of Requirements Engineering Tool Support," 32nd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO'06), Cavtat, Croatia, p. 266-273. doi: 10.1109/EUROMICRO.2006.64.

[32] Gotel O. & Mader, P. (2009). How to Select a Requirements Management Tool: Initial Steps. 17th IEEE International Requirements Engineering Conference, Atlanta, GA, USA, p. 365-367. doi: 10.1109/RE.2009.49.

[33] Young, R. R. (2004). The requirements engineering handbook. Artech House. p. 118–119.

[34] Carrillo de Gea, J. M., Nicolás, J., Alemán, J. L. F., Toval, A., Ebert, C. & Vizcaíno, A. (2011). Requirements Engineering Tools. IEEE Software, 28(4), p. 86-91, July-Aug. doi: 10.1109/MS.2011.81.

[35] Maxwell, J. (2012). Qualitative Research Design : An Interactive Approach / J.A. Maxwell. p. 121.

[36] Palinkas, L. A., Horwitz, S. M., Green, C. A., Wisdom, J. P., Duan, N. & Hoagwood, K. (2015). Purposeful sampling for qualitative data collection and analysis in mixed method implementation research. Administration and Policy in Mental Health and Mental Health Services Research, 42(5), p. 533-544. Springer. doi: 10.1007/s10488-013-0528-y.

[37] Myers, M. D. & Newman, M. (2007). The qualitative interview in IS research: Examining the craft. Information and Organization, 17(1), p. 2-26, ISSN 1471-7727. doi: 10.1016/j.infoandorg.2006.11.001.

[38] Ebert, C. & Paasivaara, M. Scaling Agile. IEEE Software, 34(6), p. 98-103, November/December 2017. doi: 10.1109/MS.2017.4121226.

[39] Dikert, K., Paasivaara, M. & Lassenius, C. (2016). Challenges and success factors for large-scale agile transformations: A systematic literature review. Journal of Systems and Software, 119, p. 87-108, ISSN 0164-1212. doi: 10.1016/j.jss.2016.06.013.

[40] Kahkonen, T. (2004). Agile methods for large organizations - building communities of practice. Agile Development Conference, Salt Lake City, UT, USA, p. 2-10. doi: 10.1109/ADEVC.2004.4.

[41] Kasauli, R., Knauss, E., Horkoff, J., Liebel, G., & de Oliveira Neto, F. G. (2021). Requirements engineering challenges and practices in large-scale agile system development. Journal of Systems and Software, 172, p. 110851. doi:10.1016/j.jss.2020.110851.

[42] Wieringa, R. & Daneva, M. (2015). Six strategies for generalizing software engineering. Science of Computer Programming, 101, p. 136-152. doi:10.1016/j.scico.2014.11.013.

[43] Stol, K. & Fitzgerald, B. (2018). The ABC of Software Engineering Research. ACM Trans. Softw. Eng. Methodol. 27(3), Article 11 (July 2018), 51 pages. doi: 10.1145/3241743.

[44] Florath, A. (2015). rmtoo. http://rmtoo.florath.net/. Last visited 28th May 2023.

[45] ArgonDigital (2016). 2016-ArgonDigital-RM-Tool-Evaluation-Scorecard [Excel spreadsheet]. Retrieved from https://argondigital.com/resource/whitepapers/requirements-management-tool-selection-and-adoption/. Last visited 28th of May 2023.

[46] Overview of DOORS. https://www.ibm.com/docs/en/elms/ermd/9.7.0?topic=overview-doors. Last visited 29th May 2023.

A

Interview Guide

INTRODUCTION

Hi, thank you for participating in this interview. We know that we have sent you previously some information regarding the purpose of this interview but due to following the structure of the standard interview process we would like to reiterate it.

We are Nora and Katarína and we are master's students at Chalmers, studying Software Engineering. We're currently working on our master's thesis which entails conducting a case study at Ericsson to evaluate T-Reqs.

The purpose of this interview is to get a better understanding of how T-Reqs is used in practice and more specifically how you use T-Reqs. We will now start with more general questions about you and your role at Ericsson and then move towards questions relating to T-Reqs. The interview is a semi-structured interview and thus we encourage you to speak as much as you can regarding the questions we ask.

We would like to ask you if it is okay that we record this interview. The recording will only be used for the purpose of this master's thesis and will be erased after its transcription. The answers that you give here today will be stored anonymously.

MAIN QUESTIONS

Background

- Tell us a little about your role at the company (main tasks and responsibilities)?
    - How many years have you been working at Ericsson and how long have you been working in your current role?
    - What previous roles did you have and did they touch upon requirements engineering or T-Reqs?

- Tell us about your experience with working with requirements tools in the past?
    - Tell us more about the features of those tools.

RQ1: How well do the expectations of T-Reqs before its implementation, match with how the tool performs today?

- **Can you describe the expectations or goals you had for T-Reqs before its implementation (or before you started using it)?**

- **How has T-Reqs performed in meeting those expectations or goals?**

    - How has it impacted your workflow?
    - How has it impacted your decision-making process?

- **Have there been any unexpected outcomes or issues/challenges that have arisen since the implementation of T-Reqs (that you know of)?**

- **How is your experience with working with T-Reqs in terms of the collaboration with the development team?**
    - **Tell us more about how you use T-Reqs in your agile development processes.**

RQ2: What advantages and disadvantages of T-Reqs can be observed in practice, in relation to traditional requirement tools?

- **Tell us more about how you use T-Reqs in practice.**
    - How often?
    - When during your workday?

- **In your opinion, what are the main advantages of using T-Reqs?**
    - Compared to more traditional requirements tools?

- **Conversely, what are some of the main disadvantages of using T-Reqs?**
    - Compared to traditional tools?

- How do you think T-Reqs affects the quality or accuracy of requirements documentation?

RQ3: What characterizes tools such as T-Reqs in contrast with more traditional requirements tools?

- **Can you describe some of the key differences between T-Reqs and the more traditional requirements tools you have used?**

- **In your opinion, what do you think are the key features or functionalities of T-Reqs?**

- How do you think T-Reqs has facilitated collaboration between different parties (SMs, developers, testers) at the company?

- **Do you think T-Reqs or tools like it will become more widely adopted in the future, and if so, why?**

CLOSING QUESTIONS AND STATEMENTS

- Do you think there are any relevant aspects that we failed to touch upon?

- Who else would you recommend that we interview?

- Thank the interviewee for their time and valuable insights.

- Explain the next steps in the research project:
    - Ask permission to follow up with a survey.
    - Ask permission to contact the interviewee about further questions or clarifications of the responses given.
    - Explain the timeline for dissemination of findings.

- Thank them again.

B

Survey

Evaluation of Text-Based Requirements Engineering Tools: An in-depth case study at Ericsson

Your participation in this survey is entirely voluntary, and all responses will be kept strictly confidential. Your individual responses will not be associated with your identity in any way. The given estimation time for the completion of the survey is approximately 7 minutes. Your input will be invaluable in evaluating T-Reqs and understanding challenges and usage of the requirements engineering tool.

Your valuable perspectives will contribute directly to the success of this study and assist in characterization of T-Reqs for the benefit of Ericsson and the requirement engineering research community. We appreciate your time and willingness to participate in this evaluation.

Should you have any questions or concerns, please do not hesitate to reach out to Audur Katarína Theodórsdóttir at audur.katarina.theodorsdottir@ericsson.com. Thank you in advance for your valuable contribution.

1. What's your current role at Ericsson?

- ◯ General system manager
- ◯ Team system manager
- ◯ Other

2. Have you used a requirements engineering tool besides T-Reqs?

- ◯ Yes
- ◯ No

3. IT technology experience

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree
I'm comfortable working with Git	◯	◯	◯	◯	◯
I'm comfortable working with the Linux command line	◯	◯	◯	◯	◯
I'm comfortable working with PlantUML	◯	◯	◯	◯	◯

4. What is your sentiment towards the following statements?

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree	I don't know
It is easy to update requirements in T-Reqs	◯	◯	◯	◯	◯	◯
It is clear **when** requirements should be **written** in T-Reqs	◯	◯	◯	◯	◯	◯
It is clear **when** requirements should be **updated** in T-Reqs	◯	◯	◯	◯	◯	◯
It is clear **who** should **write** the requirements in T-Reqs	◯	◯	◯	◯	◯	◯
It is clear **who** should **update** the requirements in T-Reqs	◯	◯	◯	◯	◯	◯

5. What is your sentiment towards the following statements?

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree	I don't know
It is easy to search for requirements in T-Reqs	◯	◯	◯	◯	◯	◯
It is easy enough to find in which release a requirement was implemented	◯	◯	◯	◯	◯	◯
It is easy enough to go back to a certain release and see which requirements are supported there	◯	◯	◯	◯	◯	◯

6. What is your sentiment towards the following statements?

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree	I don't know
Requirements residing in different repositories is a challenge	◯	◯	◯	◯	◯	◯
It is hard to keep the quality of requirements in your product domain consistent with the requirements in other product domains	◯	◯	◯	◯	◯	◯
Requirements reviews help with keeping the quality of the requirements consistent	◯	◯	◯	◯	◯	◯
I look at previously written requirements to make sure requirements are consistent and of similar quality	◯	◯	◯	◯	◯	◯

7. What is your sentiment towards the following statement?

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree	I don't know
I believe T-Reqs facilitates collaboration between gen SM and team/team SM in regards to writing requirements and their management	◯	◯	◯	◯	◯	◯

8. Please elaborate on your answer to the above statement

9. What is your sentiment towards the following statements?

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree	I don't know
It is easy to setup T-Reqs	◯	◯	◯	◯	◯	◯
It is easy to use T-Reqs editor	◯	◯	◯	◯	◯	◯
It is easy to use T-Reqs Web	◯	◯	◯	◯	◯	◯
It is easy to use T-Reqs Viewer	◯	◯	◯	◯	◯	◯
Using git for version control reduces my fear of losing work	◯	◯	◯	◯	◯	◯

10. **For me**, it's an advantage that T-Reqs...

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree	I don't know
...is integrated in the development environment	◯	◯	◯	◯	◯	◯
...is built upon standard IT technology (e.g. git, Linux)	◯	◯	◯	◯	◯	◯
...doesn't contain unnecessary features	◯	◯	◯	◯	◯	◯
...can dynamically render an HTML preview of the document	◯	◯	◯	◯	◯	◯
...has a lower cost than licensing tools	◯	◯	◯	◯	◯	◯
...is developed in-house which enables improvements of the tool via change requests	◯	◯	◯	◯	◯	◯
...enables easy synchronization with code and test cases (keeping requirements release ready)	◯	◯	◯	◯	◯	◯
...enables	◯	◯	◯	◯	◯	◯

easy tracing	○	○	○	○	○	○
...enables easy version control	○	○	○	○	○	○
...enables easy baselining	○	○	○	○	○	○
...enables automated test reporting	○	○	○	○	○	○
...allows multiple users to have access to the requirements (as opposed to limited amount of licenses)	○	○	○	○	○	○
...enables sim ultaneous update of requirements /source code/tests	○	○	○	○	○	○
...enables change tracking in Git to see who changed what	○	○	○	○	○	○
...reduce fear of making mistakes/losi ng work by being integrated with Git	○	○	○	○	○	○
...integrates with the review process (Gerrit)	○	○	○	○	○	○
...allows for easy access to information regarding the requirements	○	○	○	○	○	○

11. **For me**, it's a disadvantage that T-Reqs...

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree	I don't know
...has a high degree of freedom (low restriction of what to write in a requirement)	○	○	○	○	○	○
...misses certain functionality	○	○	○	○	○	○
...is built upon standard IT technology (e.g. git, Linux)	○	○	○	○	○	○
...is integrated in the development environment	○	○	○	○	○	○
...requires the user to fill out tags and attributes	○	○	○	○	○	○
...tool setup is difficult	○	○	○	○	○	○
...contains usability issues regarding PlantUML layouting	○	○	○	○	○	○
...lacks visualization of how requirements are connected	○	○	○	○	○	○
...improvements/bug fixes take a long	○	○	○	○	○	○

time to implement						
...contains usability issues regarding resolving merge conflicts in Git	◯	◯	◯	◯	◯	◯
...does not properly support multiple repositories (mapping & different WoW)	◯	◯	◯	◯	◯	◯
...lacks guidelines regarding what to write in the requirements	◯	◯	◯	◯	◯	◯

12. Out of the following, which are the **main** advantages of T-Reqs

- [ ] Integrated in the development environment
- [ ] Built upon standard IT technology (e.g. git, Linux)
- [ ] No unnecessary features
- [ ] Can dynamically render an HTML preview of the document
- [ ] Lower cost of the tool compared to licensing tools
- [ ] Improvements of the tool via change requests is possible due to in-house development
- [ ] Easy synchronization with code and test cases (keeping requirements release ready)
- [ ] Easy tracing
- [ ] Easy version control
- [ ] Easy baselining
- [ ] Enables automated test reporting
- [ ] Allows multiple users to have access to the requirements (as opposed to limited amount of licenses)
- [ ] Allows multiple users to work on the same requirements artefacts simultaneously
- [ ] Easy to simultaneously update requirements/source code/tests
- [ ] Change tracking in git (who changed what)
- [ ] Reduced fear of making mistakes/losing work
- [ ] Integration with the review process (Gerrit)
- [ ] Easy access to information regarding the requirements
- [ ] The high degree of freedom (low restriction of what to write in a requirement)

13. Out of the following, which are the **main** disadvantage of T-Reqs

- [ ] The high degree of freedom (low restriction of what to write in a requirement)
- [ ] Missing functionality
- [ ] Built upon standard IT technology (e.g. git, Linux)
- [ ] Integrated in the development environment
- [ ] Filling out tags and attributes
- [ ] Tool setup
- [ ] PlantUML layouting
- [ ] Lack of visualization of how requirements are connected
- [ ] New functionality/bug fixes take a long time to implement
- [ ] Resolving merge conflicts in Git
- [ ] Multiple repositories (mapping & different WoW)
- [ ] Lack of guidelines regarding what to write in the requirements
- [ ] Other

14. What is your sentiment towards the following statements?

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree	I don't know
The collaboration between gen SMs and team/team SMs is an issue/can affect the quality of requirements	○	○	○	○	○	○
The culture around perceiving requirements as cumbersome is an issue/can affect when requirements are being written	○	○	○	○	○	○
T-Reqs's disadvantages are not critical as Ericsson continuously improves the tool	○	○	○	○	○	○
It is an advantage to let the teams handle the requirements, i.e. to push the responsibility of the requirements to the teams	○	○	○	○	○	○

15. What affects the quality of requirements more

- ◯ The requirement engineering tool
- ◯ The WoW
- ◯ Both
- ◯ Neither

16. To what extent do you agree with the sentiment in the statement provided?

    The more it is the teams' responsibility to update the requirements, the better it is that the requirements are stored in the repository. But if it's the Gen SM's responsibility who otherwise never touch the repository, then I wouldn't say that they have the same benefits of the tool as we have. So from an organization point of view, it really comes down to who is working with the requirements and creating them if it's good a benefit with T-Reqs or not.

Strongly disagree	Disagree	Neutral	Agree	Strongly agree	I don't know
◯	◯	◯	◯	◯	◯

17. Which of the following would you consider to be the **main characteristic(s)** [unique features] of T-Reqs compared to traditional requirements engineering tools?

- [ ] Connected to version control
- [ ] Developed in-house
- [ ] Different views and plugins
- [ ] Ease of use
- [ ] Text-file-based
- [ ] Traceability
- [ ] Import & export
- [ ] Simple automatic quality checks
- [ ] Integrated into product repository
- [ ] Automatic test reporting
- [ ] Facilitates multiple simultaneous users
- [ ] Other

18. Which of the following would you consider to be the **key features** [most valuable features] of T-Reqs?

- [ ] Connected to version control
- [ ] Developed in-house
- [ ] Different views and plugins
- [ ] Ease of use
- [ ] Text-file-based
- [ ] Traceability
- [ ] Import & export
- [ ] Simple automatic quality checks
- [ ] Lightweight
- [ ] Modelling
- [ ] Integrated into product repository
- [ ] Automatic test reporting
- [ ] Facilitates multiple simultaneous users
- [ ] Other

C

ChatGPT-4 prompt

Okay so we are evaluating T-Reqs as a requirements engineering tool, partly by conducting interviews/survey/workshop but also by comparing the features of T-Reqs to other requirements engineering tools. This is common practise in the industry and "criteria lists" (lists of features that a requirements engineering tool can possess) are usually created to be able to compare tools to the same standard. One paper merged different criteria list and recommended a new one that is pasted below. Can you summarize this list to more high level, important features of a requirement engineering tool?

I will send it in 6 messages, one for each high level area.