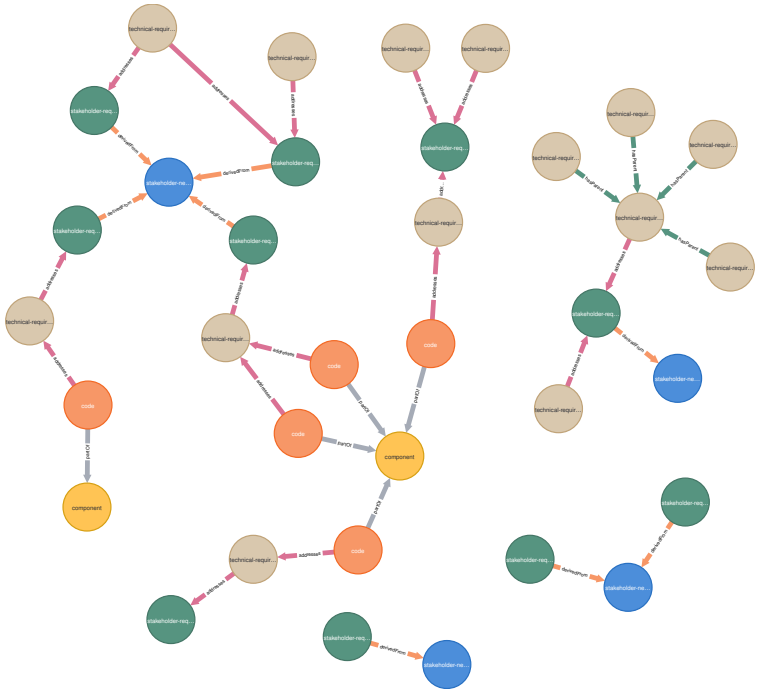
CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG



Incorporating Requirements Management into DevOps

A Field Experiment at Combitech AB using T-Reqs

Master's thesis in Computer science and engineering

Abdullatif AlShriaf
Oscar Wallin

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2024

Master's thesis 2024

Incorporating Requirements Management into DevOps

A Field Experiment at Combitech AB using T-Reqs

Abdullatif AlShriaf
Oscar Wallin

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2024

Incorporating Requirements Management into DevOps
A Field Experiment at Combitech AB using T-Reqs
Abdullatif AlShriaf
Oscar Wallin

Supervisor: Eric Knauss, Department
Examiner: Hans-Martin Heyn, Department of Computer Science and Engineering

Master's Thesis 2024
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A color-coded graph depicting different requirement elements and their interconnections, automatically generated by T-Reqs.

Typeset in LaTeX
Gothenburg, Sweden 2024

Incorporating Requirements Management into DevOps
A Field Experiment at Combitech AB using T-Reqs
Abdullatif AlShriaf
Oscar Wallin
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Context: Integrating Requirements Management (RM) into Agile and DevOps practices can address the lack of documentation and feature tracking in many agile projects.

Objective: This study examines the implementation of RM within an agile workflow at Combitech AB, aiming to identify challenges, benefits, factors affecting developer acceptance, and propose effective adaptations.

Method: Using action research, two intervention cycles were conducted with a software development team at Combitech, focusing on incremental RM adoption. Qualitative methods, including workshops and expert consultations, gathered data on its impact on the DevOps workflow.

Conclusion: RM improves traceability and communication between developers and stakeholders but faces challenges like perceived intrusiveness and the need for upfront investment. Simplified RM practices and better requirement visualization enhance acceptance. Despite initial challenges, RM integration offers long-term benefits in communication and documentation, especially considering project complexity, team dynamics, and stakeholder engagement. The findings provide insights and strategies for successful Agile RM implementation.

Keywords: Agile, Requirements Management, DevOps, T-Reqs, Software Development, Action Research, Traceability

Acknowledgements

We would like to express our sincere gratitude to several individuals and groups who have played instrumental roles in the success of our research project. Firstly, we extend our heartfelt appreciation to our supervisor, Eric Knauss, for his invaluable guidance, support, and expertise throughout the duration of this project.

We are also deeply thankful to Ingvar Andersson from Combitech for facilitating our research and providing us with essential insights and resources. Furthermore, we extend our thanks to the entire team at Combitech for their cooperation and assistance during the course of our work.

We would also like to extend our gratitude to each and every member of our thesis group for their support and insightful discussions during our meetings throughout the duration of this thesis.

Additionally, Abdullatif would like to extend special thanks to his wife Sarah for her unwavering support and understanding throughout this endeavor.

Last but certainly not least, we would like to acknowledge and thank our families —our parents, and siblings— for their endless encouragement, understanding, and patience throughout this journey.

Each of you has played a crucial role in shaping our research and making this endeavor possible. Thank you all for your invaluable contributions and support.

Abdullatif AlShriaf, Gothenburg, June 2024
Oscar Wallin, Gothenburg, June 2024

"*All is flux, nothing stays still.*"

- Plato

Contents

1
Introduction

As software engineering and development practices increasingly sway toward approaches stemming from agile workflows and methods, less time and importance is given towards achieving an extensive construction of requirements since the value-driven approach puts more importance on fast feature iterations [31]. Most modern projects only focus on listing and tracking functional requirements, typically through the use of user stories or use-case diagrams which outline the features that the product needs to include. Since agile practices focus on quick change between sprints and improved collaboration between developers, stakeholders and end-users, combining this with the more plan-driven approach of Requirements Engineering (RE) may seem counterproductive to developers [16].

However, the approach behind agile feature tracking can bring problems that a proper focus on RE could alleviate. This includes difficulties when tracking new feature changes to direct stakeholder value and having less detailed descriptions for features that can then compromise the final implementation. RE is instead well suited to fix these issues and also tends to do well within large-scale projects which comprise most of the work any tech company handles in this modern era [13]. If this approach can then be incorporated and adjusted to work efficiently within an agile workflow, the result can be a development tool that clearly shows the correlation between proposed features and stakeholder value and one that also further facilitates the constant change in an agile development process.

The study aims to investigate the challenges and benefits of incorporating Requirements Management (RM), with it being a subset of activities within the general RE effort, into an already established agile workflow, particularly within a DevOps pipeline. T-Reqs, a text-based requirements management tool designed for this purpose, will be used to facilitate Requirements Management (RM) within agile-based workflows. Additionally, the study seeks to identify specific factors critical for successfully implementing this change in the development cycle, with the goal of generating findings that can be generalized to other settings within software development.

1.1 Statement of the Problem

The RM process is often disregarded within agile software development cycles in favor of faster but less detail-oriented approaches that structure the requirements of software projects which can be due to several reasons.

Traditional RM presents challenges when integrated into an agile setting [18]. Though the final results of the approach are clear, due to the fast-paced setting, and with traditional RM being both more rigid and requiring more time to prepare, many software projects opt to not use it during development. Due to this, the process of being able to create and manage adaptable requirements that are easy to configure and change during development is of great importance and would provide value to both developers and stakeholders. However, the research within agile approaches to requirements engineering is lacking. While there are tools being developed [8] and papers outlining the different hurdles of the approach [21], there is a need for further study surrounding how to best incorporate this change within an established and real-world environment.

Communicating and formulating requirements between different stakeholder and developer roles is difficult due to several factors [25]. Companies usually work with different teams in charge of different functionalities being built in a shared project [28]. Each developer within these teams has different levels of experience with RM and may lack the ability to accurately communicate the needed requirements for a project and how they relate to stakeholder value. There are several ways to combat these shortcomings, such as having engineers dedicated to RE who manage, update and track the regularly changing requirements during the development cycles [20]. This solution can however be subject to becoming a development blocker that delays features from being worked on since only a limited number of developers would be in charge of handling the entire requirements process. Another solution is to have each team formulate and communicate their requirements to then combine these within the project. While this would potentially make for faster development cycles, the overall quality, value tracking and end-to-end traceability of these requirements would suffer due to engineers having different levels of experience and ways of formulating their concrete implementations into clear project requirements.

No solution is the best for every circumstance and setting, but there is value in finding what kind of factors project managers need to evaluate when deciding on how the RE process should be defined in their ongoing or future projects.

Tools for agile RE and RM support and automation need performance evaluation, clear specifications and a required set of features that are crucial for future tool development. Compared to the traditional way of conducting RE, its agile form avoids the pre-analysis phase and pre-planning focus in favor of conducting the RE and RM phase in parallel with development [29]. While there are tools currently in use to facilitate agile RM such as T-Reqs at Ericsson [19] and other projects such as Doorstop [8] and Jama Software [3], there is a lack of distinct performance

evaluations that pertain to how these products perform in established and large-scale workflows and projects. For companies to consider conducting and incorporating agile RE practices into their existing workflows, there needs to exist more studies that clearly outline the effects of deploying such tools inside real-world software development pipelines.

1.2 Purpose of the Study

The purpose of the experiment and study is to explore and evaluate the integration of RM into an already established agile setting that has successfully integrated DevOps workflows. We aim to explore the proposed setting at Combitech to understand the various specifications needed to successfully integrate agile RM into such a workflow through conducting two different research phases that aim to both evaluate and subsequently adjust the process and design of our change in the development pipeline. Our goal is to identify the challenges, benefits and factors of introducing an RM integration into the development process. Particularly, we aim to determine whether the change provides substantial value to a company and whether the study can yield any general practices or frameworks for integrating agile RM. While the nature of a field experiment lends itself to poor generalization, we intend to produce a final result that can both be beneficial for researchers conducting similar experiments and for practitioners looking to introduce similar types of integrations into their development workflow.

1.3 Significance of the Study

This study explores the practical application of RM within a modern agile setting, incorporating a mature DevOps pipeline — representing current trends in software development. The research identifies factors influencing the adoption of RM in this context and evaluates the value of contemporary RM tools in supporting DevOps workflows. By addressing challenges and enhancements, the study aims to provide actionable insights for optimizing RM integration in sophisticated software development ecosystems.

1.4 Research Questions

To help evaluate, refine and investigate the effects of incorporating requirements engineering within an existing and well-established DevOps workflow, we formulate the following research questions:

RQ1: What are the challenges and benefits of introducing requirements management as it pertains to software developers when integrating it into a pre-established DevOps workflow?

RQ2: What factors influence developer acceptance or resistance to incorporating requirements management into a DevOps framework?

RQ3: How can the existing agile processes be adapted to support requirements management efforts?

RE and RM are helpful to ensure that the software process results in the desired artifacts for the stakeholders. It can however require preparatory effort that might impact the fast-paced nature of agile when paired with DevOps. Our aim is to minimize this overhead through our choice of a minimalistic model, a lightweight tool and careful application of the implementation to ensure that the overarching workflow does not change in a significant manner.

Our research questions aim at identifying and quantifying the biggest challenges and benefits mainly from the perspective of the SE processes pertaining to the developers in charge of delivering a product to stakeholders. The study is contained within the requirement management process and is less focused upon the analysis as we do not aim to manipulate any process within the elicitation and analysis process of RE. Instead, we look to implement a tool that headlines the management process to then examine its effect. Though we do not manipulate any part in the analysis process, the tool and model will be designed according to the existing requirements analysis process with a focus upon finding a generalizable common ground for how you design a meta model that adheres to the existing agile workflow.

1.5 Data Anonymization

Due to the potential security risks and based on the requests made at Combitech, the data presented within this thesis will hide certain characteristics to make sure that no specifics about the project(s) are able to be deduced or shown. This includes using generic model element type names and not including the description or filename of the different requirements. We will be collecting the number of the connections i.e. links both incoming and outgoing, and for the sake of anonymity, the specific types of these links will be omitted and only the number will be preserved. With this said, these restrictions will not hinder our planned metrics collecting from being useful, as the data needed for our study to be useful only pertains to those that are independent of any anonymization efforts.

1.6 Thesis Outline

The remaining parts of this thesis will include the following:

Background will describe the background and related work concerning the different aspects of agile RE and RM to provide the reader with a thorough look at the problem space chosen for this study.

Research Method includes the different methodologies, techniques and tools chosen for this study along with the case description that is examined in order to answer the stated research questions.

Results will describe the findings from all parts of the study and also answers each research question in detail.

Discussion will analyze, evaluate and elaborate further on the findings from Chapter 4 along with describing examining the generalizability and validity of the study.

Conclusion delivers a final deliberation on the study along with potential future work based on the results.

2
Background

This chapter presents the literature review made for the study which involves the challenges of agile RE and RM, the important factors of automation and compliance, along with an introduction and background to the tool and requirement modeling used within the field study.

2.1 Agile and Automation

For any tool or process to have a chance at being successfully integrated into an already-established environment, the first factor to consider is how the environment context may require the new artifact to change to fit within its constraints or specifications. This study is situated within an agile environment, considering DevOps is often mentioned as an evolution of the agile workflow. As the term is often used quite loosely and its definition is seemingly subjective at times, studies have been conducted to properly find and formulate its characteristics. A specific one made by Lucas Gren and Per Lenberg delivers the definition "responsiveness to change" [11]. As such, the changes needed for an agile RE approach include reducing the overhead brought down by the traditional methods and designing requirements to be adaptable to changes brought on by other agile processes.

One way of trying to address these needs can come through using automation or by combining automation tools and manual developer insight. In a study made by Bjarkadóttir and Heiðarsdóttir, the tool T-Reqs was used to evaluate and analyze the use of automation within an agile setting, the process then resulted in a set of guidelines that outline the specific problematic areas that can be alleviated through the use of automated requirements feedback [6]. These areas include change history, interdependencies, differences in text language and code architecture/behavior. All of these areas bring about scalability issues, a core problem already found when combining requirements engineering practices into companies working with varying degrees of agile workflows [17]. The problems and proposed solutions often rely on a high level of context knowledge in order to reap the benefits of both approaches which goes against the value-focused and less plan-driven agile practices. Hence, there is a great need for approaches or research that can adequately produce more general solutions for combating these regularly occurring challenges, preferably those that involve agile-friendly automation of RE.

2.2 Known Issues in Agile RE

In modern software development, an agile work process has become the norm for the vast majority of companies based on its quick turnaround and close connection to shareholder needs. As mentioned before, this method of intensely focusing on feature shipping comes at odds with the traditional RE philosophy of requiring extensive and detailed pre-analysis. To accurately decipher which challenges need to be overcome to gain the benefits of RE and improved feature traceability, numerous studies within real-world scenarios have been devoted to finding the intricacies of Large-Scale RE.

In this paper, we mention that the definition of agile can vary depending on the implementation of it. The same situation is found with agile RE since the requirements process can differ not only between companies that both work in an agile workflow, but they can also look different between the various teams in a single company. In a mapping study published in 2015, the authors found that the definition of agile RE is indeed vague, though the main benefits of the approach included lower overhead costs, better understanding of requirements and reduced situations where resources are over-allocated [12].

While the benefits behind agile RE would suggest that implementing it would seem advantageous to companies, the conflict between extensive planning and frequent feature delivery has posed challenges to adopting such an approach. From a multiple case study published in 2017, challenges could be found in multiple areas of development such as communication and the traceability being correctly maintained [18]. These two areas often coincide, seeing as a lack of communication between shareholder needs and their translation to feature requirements may then result in a later change in the initial feature plan, a change that may be insufficiently communicated and subsequently changed on the test-level and other artefacts linked to that specific feature. Another issue identified in the study is prolonged feedback cycles, which involve both external and internal stakeholders. Some of these stakeholders may not follow an agile approach, or the team responsible for these features and tests may not be available to give simultaneous feedback. This issue can be alleviated by providing a clear context of the project and what is needed of each member of the team while also providing effective communication channels.

Moreover, the study brought up that several participants did not deem user stories and tests to be sufficient as a documentation alternative. Inayat et al. further expand on this form of minimal documentation, where the aim of concise and specific user goals in user stories is very vulnerable to a system that regularly changes through either requirements, project stakeholders being unavailable or a change in project complexity [14]. The assumption behind agile practices is often based on having either on-site customer access or assuming that relevant users are available at any time. However, this is rarely the case and can result in several cases where a question about an unclear user story or feature becomes a backlog item for a future sprint rather than being quickly explained and fixed [24]. Providing these feature

requests more clearly and well-documented while still sacrificing the extensive pre-planning of traditional RE would help both parties, seeing as the customer can more easily validate that the feature is correctly communicated while the developer requires less intercommunication to create the underlying implementation and artefacts. Through these multiple studies, we can deduce that bridging the gap between traditional, plan-driven RE and agile practices can be of interest to many companies.

In an extension of the same study published in 2021, both challenges and possible solutions were expanded on [17]. Of particular note, the challenges in the areas surrounding the long-term maintenance and understanding of the system, a process that adequately supports change and evolution as well as the representation of requirements are of interest for this paper. Since the value chain is spread across several different actors with varying experience within RE, there also exists the difficulty of communicating features into requirements and translating those across different teams. This is further exacerbated if each agile workflow differs between teams and if their backlog process or requirements elicitation phase does not work in the same way. Savolainen et al. further note the lack of dedicated tools that would help with these problems and describe the 'rediscovery' of RE in agile development as needing a mixture of both old and new practices along with incremental introductions of these new implementations [27].

The struggle of efficiently integrating RE into agile workflows presents a set of challenges due to partly inadequate tool integration and limited support for concurrent changes among the teams. A proposed solution strategy by A.P. Muhammad et al. is to utilize tools that allow developers to take ownership of requirements and pair it with regular meetings with customers to help align the visions of stakeholders. This strategy gives a clear picture of the value of the product for the client and how the team contribution is viewed. Frequent feedback cycles enables direct communication with the client to explore the developer's interpretation and application of high-level requirements to maximize the benefit of the RE process [22].

Establishing a shared understanding within the process of RE presents various challenges and considerations. In agile development, striking the right balance of documentation is essential, as excessive or insufficient documentation can lead to issues. While agile methodologies typically advocate for less documentation, this approach may overlook crucial details, such as the rationale behind requirements. Moreover, industries like automotive and communication, with stringent compliance requirements, demand comprehensive traceability across all levels of requirements, tests, and code. This poses a significant challenge for teams as these artifacts evolve concurrently with the corpus of requirements and legislation. Teams encounter requirements in diverse formats and levels, ranging from textual requirements to user stories, across system and software levels. Typically, they initiate with higher-level requirements and subsequently derive lower-level ones, although tracing these requirements can prove challenging in large, complex systems. To streamline this process, the integration of tools aligned with an agile requirements strategy is indispensable, facilitating parallel work for multiple teams [22].

By identifying these different challenges and contextualizing them within our scope of introducing a software tool within a pipeline, we can capture several characteristics needed in order for such an agile RE implementation to be successful or appealing to SE companies:

- The tool needs to be modular and flexible and capable of fitting into any type of agile setting that handles RE. Since documentation and traceability can happen in many different ways (dedicated text files, linked test cases, linked backlog items, linked sprints, etc.), the team needs to be able to tailor the tool based on their interpretation and implementation of RE.

- The output or product that represents traceability needs to be easily interpretable such that any developer working in an agile setting can understand *what* the tool does and helps with, *how* the tool documents and/or visualizes the traceability along with how they can interact and expand their use of the tool.

- The tool needs to be slotted into an existing agile framework through several iterations, gradually adding value to traceability to both developer and customer while still maintaining the fast-paced and high-value nature of an agile workflow. This includes clear intercommunication standards, documentation that is available when personnel are not and the software tool being able to fit within a CI/CD pipeline structure.

2.3 Requirement Modelling for Traceability

Traceability ensures that each requirement, design element, and test case is properly linked and traceable throughout the project life-cycle. This helps teams verify that all requirements are met, facilitates impact analysis, and supports compliance with standards [30].

Traceability can be categorized into two types: Horizontal traceability, which involves tracing between artifacts of the same type, such as tracing between different requirements, and vertical traceability, which involves tracing between artifacts of different types, such as tracing between requirements and tests [26].

A Trace Information Model (TIM) is a framework used to represent and manage trace-links and relationships between artifacts created during the development of a system [10]. By defining both horizontal and vertical traceability, a TIM enables a structured approach to maintaining comprehensive traceability across various elements and stages of a project, thereby supporting effective project management and quality assurance.

2.4 RE and Compliance

Researchers [23] found three key challenges when using requirements to ensure compliance through contractual agreements, laws, regulations, or industry standards. These challenges include dealing with the size and nature of regulatory text, navigating contractual complexity, and addressing the sheer scale of the system. In a DevOps context, these findings underscore the importance of addressing regulatory and contractual challenges within the development workflow to ensure effective integration of requirements and compliance measures, even within projects with a smaller legal regulatory body and limited scales.

While on the topic of compliance and RE, researchers conducted a study by introducing a pattern-based method for identifying and analyzing laws, a method that was designed to then be integrated into conventional systems and software development processes [5]. The uniqueness of their approach lies in the examination of common methods employed by lawyers for law identification and analysis, with the resulting knowledge encapsulated in patterns. This pattern-based methodology is derived from the subsumption method, distinguishing it from other approaches that typically employ formal logic for the formalization and analysis of laws.

3
Research Method

The methodology adopted for this master's thesis involves conducting a supervised field experiment, drawing inspiration from action research and design [32] where we investigate the research questions in a natural pre-existing setting with some level of deliberate intrusion. Specifically, the chosen approach is the action research within a field experiment setting, consisting of two main intervention cycles and necessary preparation and evaluation.

3.1 Action Research

Action research aims to explore and subsequently solve a problem by conducting research and practice to work synergistically where the two phases inform one another [4]. The reasoning behind this approach stems from the fact that the solution for incorporating RE into agile processes can be initially unclear and its final implementation depends on the context. Our action research method aimed to test the proposed solutions in real situations within an organization that involves several people who have different work objectives and perceptions, lending to multiple factors to consider and test in order to find a final solution that fits the real-world scenario.

This method of applying feedback from previous cycles to the current one was agreed to fit within the purpose of our study, where the iterative cycles can both identify the more generalizable factors to consider while also picking upon the specifics that provide value to the real-world setting we chose to conduct our research in. Additionally, in order for action research to be relevant for other researchers interested in doing a similar study, Peter Checkland and Sue Holwell argue that *recoverability* is of utmost importance [9]. This means that anyone interested in our subject should be able to *recover* the process done during our experiment. Through the different types of data collection techniques and cycles documented in this paper, we aimed to provide this recoverability and thus improve upon the credibility of our final results.

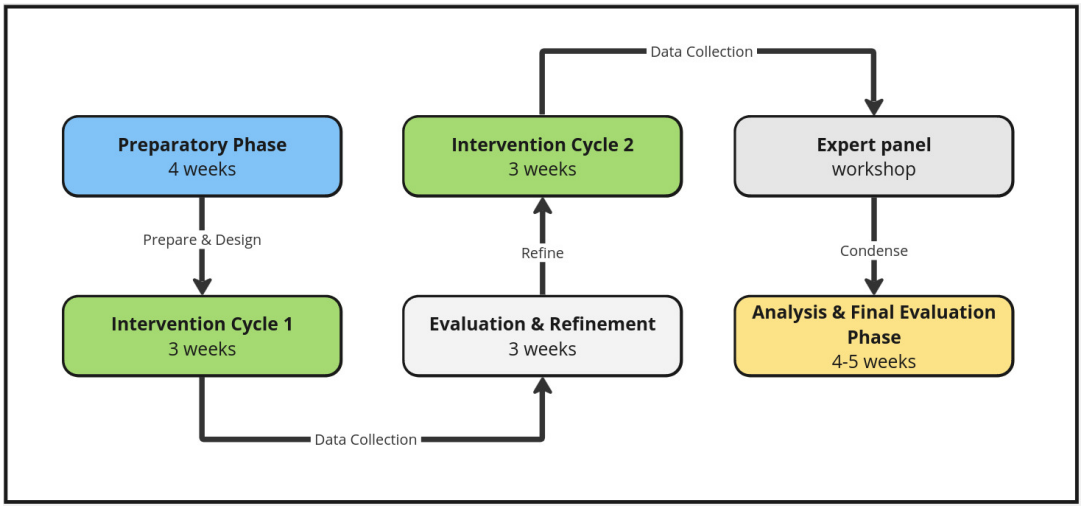


Figure 3.1: A visualization of the different stages of the study along with their respective time allocation.

3.2 Iterative Research Cycles

The study was comprised of six stages, including two intervention cycles where the customized interventions were implemented and tested within a natural environment, aiming for minimal disruption during usage. The five primary stages culminate in a discussion with an expert panel toward the end of the second intervention cycle. The timeline allocated for each stage is illustrated in Figure 3.1.

During the initial preparatory phase, surveying the existing landscape in terms of workflows and processes was important to lay the ground for the rest of the research. In this phase, our aim was to introduce the concept of RE, focusing specifically on the RM aspect, along with related workflows and tools to the study participants. In this sense, the study transforms into a supervised field experiment with an introductory effort undertaken during the initial assessment phase. Qualitative data is explored to gain insights into the existing environment: Projects, stakeholders, techniques, tooling and setting. Since the team is not familiar with the tool used during the study, a workshop is executed before the first intervention cycle for them to learn the required knowledge needed to use the tool effectively.

Subsequently, we began the first intervention cycle where change is introduced, monitored and assessed in an evaluation and refinement phase. The results from the previous stage are incorporated into this phase, necessary adjustments are made and parameters are refined based on the results of the initial intervention cycle in order to improve the final results in the second intervention cycle. We then entered the comparative analysis phase to compare the results from the previous two cycles identifying patterns and variations, along with evaluating the study as a whole.

3.3 Case Description

Combitech was chosen for this study due to its strong reputation as a consulting company with a diverse clientele and a focus on innovation and agile methodologies. The company's openness to research makes it a suitable environment for testing and implementing the study. Additionally, as part of the Saab group, Combitech has a tradition of internal compliance requirements from working on safety-critical projects.Additionally, Combitech's existing DevOps-oriented culture provides a solid baseline for assessing the impact and feasibility of introducing RM into an agile environment.

3.3.1 Selection of Team

Before commencing the study, we outlined specific a priori inclusion criteria for team selection: the team must have an established DevOps process, operate using agile methodologies, and maintain communication with stakeholders. The selected team for this study comprises individuals with diverse expertise in agile methodologies and DevOps practices. Comprising three members, the team facilitates focused qualitative analysis. With over two years of project experience, the project's maturity ensures depth and stability. The team's composition ensures the perspective of developers as stakeholders is considered from various angles within that perspective. With roles including back-end, front-end, and senior systems development, the team brings a holistic understanding of the modern developer needs and challenges, facilitating effective integration of agile RM within Combitech's existing workflows.

The mono-repo architecture of the team's project centralizes code, documentation, and other artifacts, streamlining version control and collaboration. Integrating RM within this framework presents unique challenges and opportunities, such as ensuring traceability, managing dependencies, and maintaining consistency and links between components of the project.

As in-house consultants, requirements and compliance demands originate not only from internal stakeholders but primarily from the end customer. This means that the team maintains a customer-centric focus, aligning development efforts with the needs and expectations of the end-users. This perspective ensures that RM integration addresses both internal and external requirements, enhancing product quality and customer satisfaction.

3.4 T-Reqs: A Textual Requirements Management Tool

T-Reqs (Text-based Requirements) is a tool that has seen use in an established industrial setting at Ericsson [33]. Within this study, the [1]open-source version developed in-house at Chalmers University of Technology was used during the cycles

[1]Accessible online: `https://gitlab.com/treqs-on-git/treqs-ng`

of the field experiment. T-Reqs uses a Type and Trace Information Model (TTIM) that extends the functionality of a traditional Traceability Information Model (TIM) which primarily focuses on representing trace-links and relationships between abstract trace artifacts [10]. In contrast, the TTIM specifies the different elements *and their types* to be traced along with the different types of trace links between them. These links can either be optional or required depending on the circumstance and rigidity needed for the model. Text-based elements are usually documented in markdown files but any element can be housed within programming language syntax, making it possible to keep track and trace features between multiple languages within the same project. Each element and its possible links are characterized by different XML tags that describe the different attributes related to the requirement.

Since T-Reqs is a static analysis tool that operates on the specified TTIM which can specify any number of rules and elements, much of the onus is on the users of the tool to create a model that sufficiently maps the traceability while also not making the number of elements or links too complex.

The a priori selection of T-Reqs as the RM tool for this study is supported by its status as an open-source tool under active development. This enables quicker resolution of emergent bugs and facilitates customization and adaptation of the tool to suit the specific needs of the study. Additionally, the open-source nature of T-Reqs allows for the incorporation of new features to be experimented with during intervention cycles. This flexibility and responsiveness contribute to the effectiveness and suitability of T-Reqs for the research study.

T-Reqs being a Command-Line Interface (CLI) tool means that it can be containerized, incorporated into DevOps pipelines, and seamlessly integrated into other scripts for automation, metrics gathering, and custom script-based checks which is vital for the chosen methodology of our study. This versatility allows T-Reqs to be efficiently utilized within modern software development workflows, enhancing automation and facilitating the integration of requirements management into various stages of the development process.

3.5 Data Collection

As per the outlined research methodology, data was collected in the initial assessment phase and throughout the observation phase during and at the end of the first and second intervention cycles.

A qualitative data collection approach was used in the initial assessment phase to help guide the design of the first intervention cycle. This entails observing the overall workflow, joining meetings and analyzing the pre-existing code found in the project, as well as reviewing any available documents and artifacts that outline the setting. The domain knowledge captured here was then used to customize the tool to optimize performance during Intervention Cycle 1 (ICI).

Apart from the document reviews, these approaches to collecting qualitative data were continuously used throughout the study to gain more insight into our implementation while also observing any changes that the implementation naturally brings. These changes were not manipulated by ourselves and the only other direct intervention done during the cycles was to answer any questions or address potential bugs or problems with the tool that we introduced.

We categorize the different techniques based on the data collection phase as follows:

- *Preparatory:*
  - Document Review: Analyzing and reading the current documentation from the point of view of developer new to the project.
- *Quantitative:*
  - System Metrics: Specific data was collected throughout the study, both from the CI/CD pipeline as well as other project management tools currently used by the team.
- *Qualitative:*
  - Reflection meetings: Team-wide discussions planned after each intervention cycle to assess and evaluate the effects of the study in-depth.
  - Observing the development process: A continuous process to evaluate the current agile processes, their changes concerning RM during the study and how RM can efficiently be integrated into an existing agile workflow.
  - Discussion with an expert panel: This session captures insights and perceptions, revealing effectiveness of the interventions, usability of findings, and emergent patterns of the RM application in real-world settings. Experts provide recommendations and validation for refining RM strategies based on observed outcomes.

3.5.1 Document Review

As a first measure, all types of documentation in the project repository were reviewed and analyzed. This involves artefacts ranging from retrospectives and technical documentation to source code, comments and configuration files. The purpose behind this is to approach the project from the point of view of a developer recently introduced to the team and the project and then ask the following questions: How does the existing documentation illuminate the different services or parts of the project? Can developers understand the features of each component and comprehend the relationships between them? This review measures the team member interchangeability of the project, and can also be loosely connected to its maintainability. Lastly, this document review can also shed light on how to most efficiently model the project from an RM perspective, along with how the types in the model will be described and created.

3.5.2 System Metrics

During the planning phases for the intervention cycles, special metrics were put in place to automatically collect data relevant to the system under study, such as specific requirements-related metrics in the DevOps pipeline. This type of data is insightful as it provides a quantitative source of objective data points that come directly from the setting of the experiment. Special consideration on top of the laws for handling personal and company information for the sensitivity and anonymity of the collected data is of paramount importance to help ensure the reproducibility and publishing of the acquired data.

In order to obtain quantitative data on the integration and usage of RM tools and processes into the workflow, we continually collected specific metrics whenever the pipeline was used in the project. These metrics are based upon the TTIM specified in T-Reqs and include the following:

- The number of elements per type in the model
- The total number of links in the model
- The number of inlinks per type in the model
- The number of outlinks per type in the model

These metrics act as a complementary part to the qualitative data collected, showing *when* the model was used, *how* it was used during different parts of the process as well as being a part of the final artefact of the study as the metrics gathered can help to describe if RM enhances and clarifies the traceability of a project. While these metrics do not singlehandedly answer the research questions stated for this study, they will serve as a clear showcase and strengthen the validity of the experiment. They may also support the different observations made regarding the process and how it correlates to RM as it can identify during which period most high-level requirements were made and when these were linked to low-level code artefacts.

Adding to these metrics, a continuous stream of information from Continuous Integration (CI)/Continuous Delivery (CD) were extracted from the pipeline to provide insight into the number of runs and how it impacts the RM process. This information includes the performance of various checks and runs of T-Reqs, along with developer insight on which parts of the pipeline process are most valuable.

3.5.3 Reflection meetings

One form of qualitative data came through the scheduling of reflection meetings that involve the project group in focus. While the main subjects were developers close to the code, other roles such as project managers or owners were occasionally present due to their interaction with the study. These occured after each intervention cycle with each one having tailored talking points based on the main goal of the cycle along with being heavily anchored to our research questions. The first meeting after First Intervention Cycle (ICI) provided a general evaluation along with going in-depth around the following topics:

- The experiment and its impact on traceability and if the implementation is deemed valuable
- The Requirement Meta Model and its performance
- The implementation of RM into the existing agile workflow and its obtrusiveness, benefits and drawbacks
- The current tool maturity and any future improvements viable for Second Intervention Cycle (ICII)

The second reflection meeting following Intervention Cycle 2 (ICII) prioritized the results along with final evaluations that relate to our three research questions. This means that a large focus of the discussion was to see if the tool integration and benefits outweigh its overhead and challenges, along with finding the important factors that both alleviate and inhibit the integration of these types of tools. Lastly, an examination of the overhauled process was done to assess its disruption to the workflow along with further suggestions for the implementation that may inspire future work and studies.

3.5.4 Process Observation

While this is of particular relevance in the preparatory stage of the study, we aimed to observe the existing process steps as well as the changes to these steps throughout our field experiment. This means that we attended the various meetings that occured during the 3-week development cycle including daily stand-ups, backlog refinements and sprint plannings. The main focus was to not influence any choices for the developers but to answer questions if need be and most importantly find ways to better and more organically integrate RM into their existing workflow through these observations. Another observation of the existing configuration for the pipeline along with looking at historical and contemporary runs of said pipeline clarified and helped in deciding when and how to better integrate RE-related activities into the team's CI/CD workflow.

3.6 Data Analysis

Data collected in the preparatory phases for ICI and ICII yielded a set of changes that are to be introduced prior to the start of each intervention cycle, laying the ground for the team to undergo the intervention cycles.

At the culmination of each intervention cycle, the data points collected and the feedback acquired served as the input for designing decisions in the subsequent intervention. Subsequently, during the following meeting, the newly implemented solutions were discussed and evaluated. To analyze the qualitative data gathered, a grounded theory approach was employed.

This iterative method involves open coding, where raw interview data is grouped into codes. The process continues with axial coding, defining categories to organize codes, and culminates in identifying a core category that forms the basis of the derived theory. This cyclic data collection and analysis method allowed for a comprehensive understanding of evolving patterns.

4
Results

In this chapter, we discuss the results obtained by the two intervention cycles. The results of each intervention cycle can be divided into two parts: The results of the preparatory/evaluation phase and the results of the intervention itself.

At the end of the preparatory phase, the variables (setting, tools, environment) of the intervention cycle were set and stayed unchangeable for the duration of the intervention cycle.

4.1 First Intervention Cycle

ICI acted as a baseline for familiarizing the team with the concepts of RM as well as investigating how to smoothly integrate the tools and processes of RM into the agile and DevOps way-of-work for the team.

4.1.1 Preparatory Phase

In preparation for ICI, we identified three main variables to tweak, ranging from technique to process.

Project Structure

The first step of this stage was to gain familiarity with the project, involving the project structure and different working parts to make beneficial decisions when modeling the domain. This was mainly done through a low-level approach of examining the repository while also discussing the structure with the developers in charge.

The project is housed in the cloud using Microsoft Azure and incorporates a DevOps pipeline and workflow, meaning that each commit needs to pass specific pipeline checks that can not be locally run. We also found several interesting points of interest that either influenced our model or helped with the evaluation of the tool:

- **Monorepo and multiple languages:** The project is housed within a mono repo, meaning that several different services and projects are found within the same environment. This involves different microservices along with several layers of the project including backend, frontend and infrastructure artefacts. This is of vital importance to the context of the model, as it requires one that can easily adapt to new projects or services without having to reinvent

the model structure. The project also houses several different programming languages such as *Golang*, *JavaScript* and *Kotlin*. This in turn can both prove and troubleshoot different parts of the software tool T-Reqs used to map elements of the model, further evaluating and improving its real-world appliance.

- **Absence of tests:** Noticeably, the project lacks any representation of *tests*, as the team is in the process of looking for high-level tests, i.e. integration and/or system tests, which better suit the micro-service nature of the project. This does hinders the possibility of requirement validation as there are no concrete artefacts that serve to show that the project features work correctly and as specified. However, these justifications can also be made to fit the nature of the project through the form of items not directly constricted to code artefacts, either through documentation of system tests linked to certain features or other means. These types of validation techniques aim to be tested in ICII while the first cycle is focused on both accurately and efficiently mapping the project to suit the developer workflow.

- **Specialized, undocumented knowledge:** An important factor that was communicated early on by the group members was that much of the project is built on very specific libraries and functionalities, the understanding of which is not documented and is only understood by the current team members. This means that a change, addition or removal of a team member could have vast repercussions since it would result in either a large loss of knowledge or a very steep learning curve for the project. The factor of the project needing specific or specialized knowledge and having several obscure parts can serve as another measurement of how successful the tool integration has been at the end of the project. The optimal solution would then be that a new developer can see the relations and traces between different libraries and their functions to understand the different components of the project.

Requirement Domain Modelling

Modelling was done by identifying key concepts, artefacts and documents in the project scope and modelling their interactions to communicate explicit traces between the different stakeholders in the project. By examining the documents and static artefacts, an initial model was presented to the team at a preparatory workshop, see Figure 4.1. Input from multiple members indicated a desire to not have the *component* type as an intermediary link between *requirement* and *feature*, but to allow *feature*s to directly trace to a *requirement* while maintaining a mandatory link to a *requirement*. The reasoning behind this is that a *feature* implemented in multiple *component*s could address a *requirement*.

As a result of refinements to the TTIM, we arrive to the model illustrated in Figure 4.2. The model features five distinct types, where *stakeholder-need* and *stakeholder-requirement* are considered *high-level*. Here, *stakeholder-need* originates from and is formulated by the client while *stakeholder-requirement*s are formulated from discussions between the client and the team. While *stakeholder-need*s are somewhat

static, *stakeholder-requirement*s can be updated and allow for hierarchical ordering due to the reflexive trace-links *hasParent*. The next abstraction level is the *requirement* type representing technical requirements that are formulated by the team. These technical requirements can create a hierarchy due to the reflexive *hasParent* tracelink.

At the lowest abstraction level, *features* exist with two mandatory traces to *component* and *requirement*. For vertical traceability, which involves tracing between items of different types [26], the linkage from low-level types to higher-level types helps in understanding how specific features or functionalities contribute to fulfilling objectives. This understanding enables effective prioritization from the developer's perspective.

On the other hand, for horizontal traceability, which entails tracing between items of similar abstraction levels [26], we employ inverse linking — from *stakeholder-need* to the less abstract *stakeholder-requirement*. This approach emphasizes the importance of providing context and rationale for the more detailed type, i.e., *stakeholder-requirement*. A YAML representation of the requirement domain meta-model is found in Appendix A.

Automation of RM in DevOps Workflow

In an effort to leverage the team's reliance on CI/CD workflow, we wanted to augment and leverage the existing pipeline for two main goals in ICI, see Figure 4.3:

- Checking the consistency of the requirement model, i.e. whether requirements are well-formed with respect to the defined types and trace links.
- Collecting and storing quantitative metrics about the state of the model at every pipeline run.

As per the policy to store collected data in a company container before extracting it for data analysis (as described in Section 3.5.2), a special blob storage needed to be created to host the collected metrics gathered in the pipeline.

In order to integrate T-Reqs in a DevOps environment, a change was authored to allow the containerization of the tool so that it could be utilized in a pipeline with limited overhead. Containerization also allows for immutable versions of T-Reqs to be used in the pipeline throughout the intervention cycle. Finally, a containerized image of T-Reqs was built and pushed to the company's Azure Container Registry (ACR). To not be too obtrusive in ICI, a choice was made to make the proposed stage in the pipeline non-blocking, i.e. the pipeline run won't fail in case the model is inconsistent, but it will issue a warning. Appendix B outlines how the stage was implemented, note however that it contains some redacted information to safeguard the company's internal data.

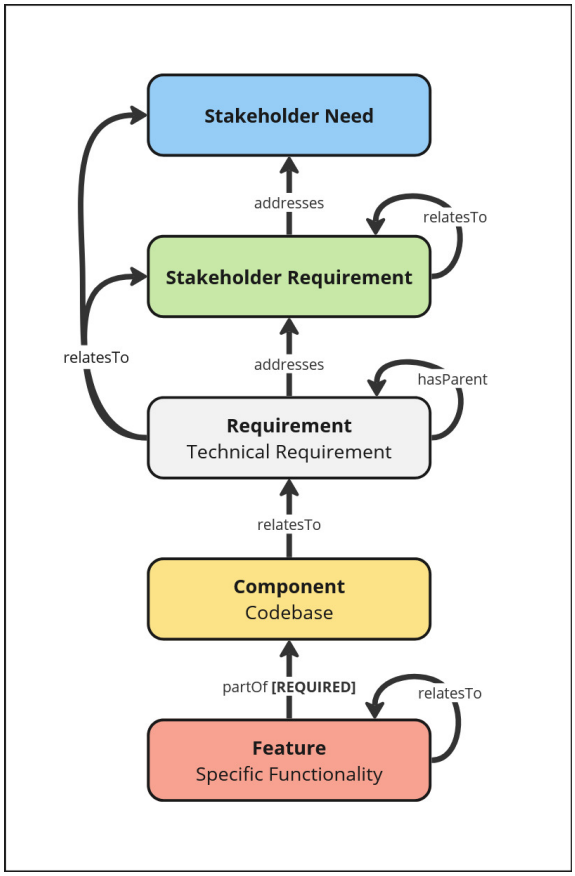


Figure 4.1: First iteration of the Type and Trace Information Model, featuring five distinct types, one mandatory trace, and three reflexive traces. In this model, a *feature* doesn't directly link to a *requirement*

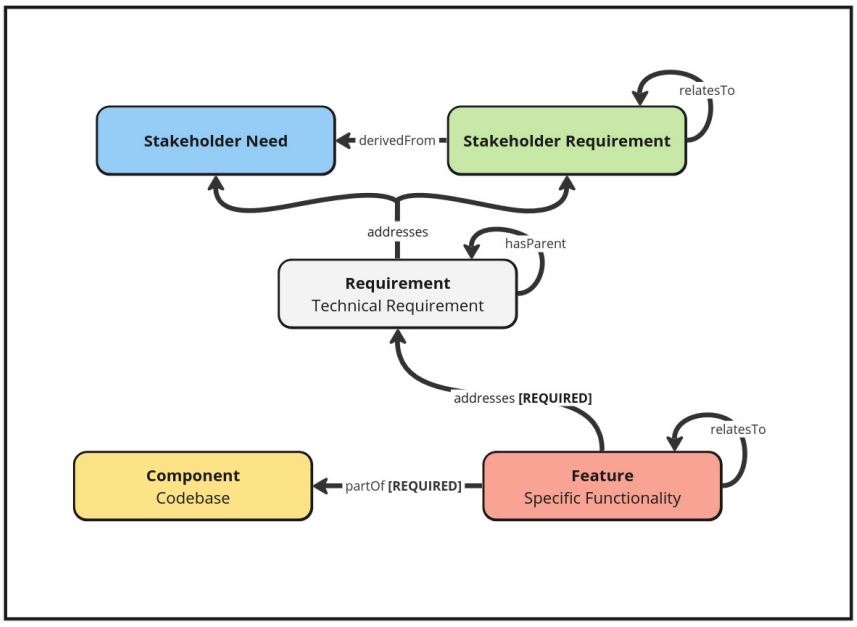


Figure 4.2: Type and Trace Information Model for the first intervention cycle, featuring five distinct types, two mandatory traces, and three reflexive traces.

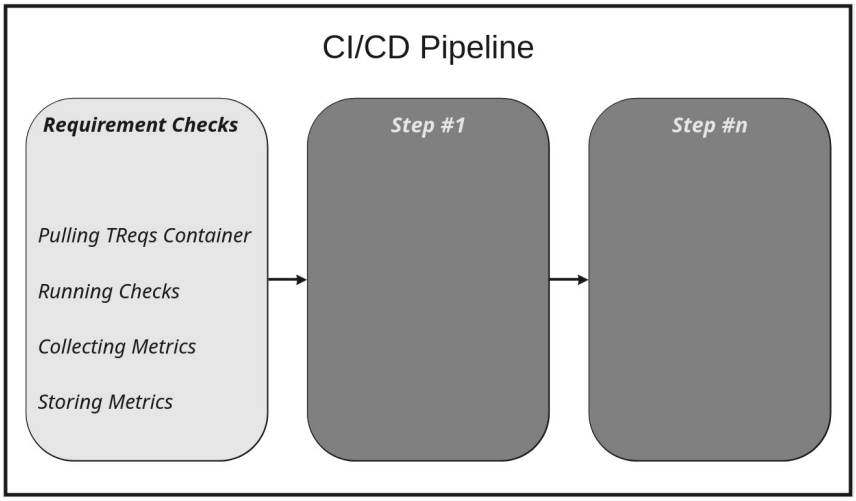


Figure 4.3: The stage *Requirement Checks* is introduced to the team's pipeline, comprising four main tasks that utilize a containerized version of T-Reqs, alongside supporting scripts for automating checks, extracting and storing metrics, and visualizing feedback within the DevOps platform.

As for the metrics, we created four scripts (see Appendix C) responsible for extracting the following metrics:

- Number of elements in model: The total number of instances of each type in the model.
- Number of links in model: The total number of tracelinks (in both directions) between the instances of the types in the model. The metric is grouped by the type of the tracelink, e.g. `hasParent` or `partOf`.
- Number of out-links per type: The total number of outgoing tracelinks from each type in the model.
- Number of in-links per type: The total number of incoming tracelinks from each type in the model.

Each link in the model is characterized by exactly one incoming and one outgoing connection (no one-to-many relationships). The final two metrics provide valuable insights into the interconnected nature of the model. The total count of incoming and outgoing links for each element type is inherently influenced by the design of the requirement model. Although the aggregated number of incoming and outgoing links is always identical for the model as a whole, the variations in these counts across different elements serve as a gauge of the effectiveness and utility of the chosen requirement model.

The metrics are produced in comma-seperated values (CSV) format for ease of analysis and manipulation. We note that the collected data does not contain any information about the contents of the types nor their location in the repository. We also recognize that some metrics are closely dependent on the topography of the TTIM.

Observations

The first item examined for our observation process was to look more closely into the specific agile workflow that the team works with. The team employs *Scrum*, but with the modification of adopting *The Cooldown Manifesto*, stating:

> "*Scrum is too fast! It has made us hurry like animals. There is (supposed to be) rules to this scrum, so we wrote us a manual, a (sort of) step by step booklet for you to get your sprint on track, not your team pushed back.*

1. Between sprints or the first week of the sprint, there is a *cooldown* period.
2. During cooldown, we have the following Scrum events:
   - Retrospective
   - Demo
   - Backlog refinement
   - Sprint planning
3. No Daily Scrums during cooldown.
4. Instead, we have the following new events:
   - Demo/Workshops with the customer.
   - Internal knowledge sharing workshops.
5. The remaining time during cooldown is dedicated to investigating and refining new user stories so they can be planned for the next sprint!
6. Time for improvements and refactoring.

This form of agile came to benefit RM in a natural way where the implementation did not feel foreign or intrusive to integrate. During the cooldown period, the team focused on addressing the bulk of high to mid-level requirement types, namely *stakeholder-need*, *stakeholder-requirement*, and *requirement*, by creating, refining, and linking them. This period is crucial as it facilitates client-developer interactions, enabling clarification on existing functionality and the proposal of new high-level requirements. The retrospective made early on in the cooldown period also makes sure that the changes and additions made to the model from the previous sprint are linked and able to be traced, thus improving the overall traceability. The emphasis on low-level types, such as *feature*s and *component*s, will occur during the next two weeks of the sprint, which is when the majority of development effort takes place. Daily SCRUM meetings will also play a pivotal role in clarifying, creating, and deriving *requirement*s. We also observed that stakeholder requirements and needs can require clarification to more clearly convert them into technical requirements and their following code functionalities, while some may be larger in scale than previously planned. The adaptability of T-Reqs and the model chosen for the project lends itself to this, as documentation of any type can be changed whenever the need arises, while the reflexive traces found in the model make it easy for evolving requirements to exist and to be traceable.

When executing the workshop that handles the introduction of T-Reqs along with answering questions during backlog refinements, sprint plannings and SCRUM meetings, we gathered a large collection of data regarding our implementation. Early on, the team members agreed on the tool being relatively easy to use. Since the tool is designed to be close to the developers and is accessible through the CLI, this confirms that the usability factor is of substantial importance when you look at the developer acceptance of RM. During our attendance in meetings and sprint plannings, discussions of future improvement were freely discussed and resulted in points that can be anchored to several factors:

- **Visualization and accessibility** - *HTML outputs and generated, easily accessible lists made on pipeline runs*
  - As the project and the types in the model gradually grow in size, the visualization of the project model needs to be scalable with this change. If the output of the model ends up with excessive lines of text, hard-to-read diagrams or formats that require prior knowledge beforehand, the work done on creating the model would be in vain. In this case, what T-Reqs outputs through CLI messages, PlantUML charts and markdown files later become harder to read and decipher, especially for stakeholders that are new to RE. This shows the need for RM tools to have more options for visualizing the model and requirements through other means. To see the value of improving this area, new ways of visualization such as an HTML report or a linked chart outlining the different traces will be implemented in the following intervention cycle. This will then be evaluated to see if it would be valuable to all stakeholders as it can both ensure the developer that the model holds, while also being an artefact that can be shown to validate the traceability.
- **Streamlining traceability and model changes** - *streamlined tracing between low-level to high-level abstraction levels, different restrictions on the same TTIM*
  - An early suggestion during the observation was that the tools that handle RM need to offer a streamlined way of tracing a low-level element through several abstraction layers in one iteration or command. This is a feature that could improve on instantly finding the value chain of one feature or component for a stakeholder to see, or for a developer to ensure that the type is linked as intended. To add to this, visualizing this path through means discussed in the last point would make it easier to isolate separate changes when e.g. describing a change done in the current sprint.

    Another suggestion was to add commands that can add, change or ignore parts of the chosen model when running a validity check to see which parts of the model go against this temporary condition. The tool used in this study is capable of this through scripts or by creating several iterations of the model but this technique impacts its perceived usability and is not intuitive. Therefore, RM tools should offer ways to ease the process of adding temporary restrictions on the current model.

While moving through backlog refinements, adding mid and high-level requirement types as well as becoming more familiar with the tool, the team also mentioned how an increased focus on maintainability had been initiated by the project manager. The relation between T-Reqs, RM and maintainability brought up two main points during our observation that would point toward RM being a complementary factor to the effort of increasing maintainability. While not explicitly stated in the TTIM for ICI, a bugfix or a project-wide effort to update deprecated libraries or components can be expressed, logged and tracked through the different types of the model, typically through the *requirement* or *feature* types in this specific case. Since the increased traceability enables the developer to quickly understand which parts may be at fault when examining a potential issue, RM can thus directly help to increase the maintainability of a project.

Another observation after implementing RM regarded the state of the documentation. As mentioned earlier, most documentation was found outside of the project and a search in the repository prior ICI would not increase the understanding of the project significantly. However, as the different types in the model were mapped during the first sprint (especially during the cooldown period) we could observe a noticeable difference in detail when looking at e.g. README's of different components or when translating user stories into more technical requirements. Since T-Reqs is mainly a tool for developers, these descriptions and improved readability directly impact those working hands-on with the code in the project as it makes for a much more detailed output when running checks in the pipeline or when examining the list derived from the TTIM.

4.1.2 Evaluation of ICI

After executing our changes and collecting the designated metrics, the following three weeks were spent analyzing the results as well as evaluating the intervention cycle with the team. The most notable and information-rich activity during this period was the reflection meeting as it brought about several interesting themes related to our research questions, many of which can be directly related to or validated through our metrics.

4.1.2.1 Reflection Meeting

The first reflection meeting had its focus on evaluating the first cycle along with finding possible improvements for the second and final cycle. The themes of each round of questions were related to a broad and general evaluation of the preceding three weeks, questions and adjustments to the meta-model, the experiment and its integration into the already existing process and lastly a look into the tool maturity and further improvements upon T-Reqs. The specific questions presented can be found in Appendix D.

The answers and discussions presented during this meeting were recorded and later transcribed, with emergent coding (executed through ChatGPT) being used to map out the different themes from the meeting and how they relate to our research ques-

tions. Note that due to both security concerns, respect for privacy and the fact that the meeting was conducted in Swedish, the full transcription will not be available in this paper.

From the meeting, we could identify several themes that were then related to each research question:

RQ1: Challenges and Benefits of Introducing Requirement Management in DevOps for Software Developers

Challenges:
Tool usage and its impact on workflow: Software developers may experience a disruption in their usual workflow when new tools and processes are introduced. This initial adjustment period can lead to resistance and a learning curve that may negatively affect productivity.

> "*Since we are so close to the development process, we often think in terms of user stories when writing requirements which then sort of results in 'double bookkeeping'. The question then becomes - within agile RM, who should set or translate the requirements? [...] The responsibility for the backlog often falls to the product owner within an agile team, so there could be some danger with moving the elicitation of requirements onto us developers.*"

Technical and organizational challenges: The implementation of RM tools like T-Reqs may pose technical obstacles, such as integrating XML into the codebase, which could reduce code readability and increase maintenance complexity.

Advantages:
Historical and future traceability of requirements: Implementing RM can enhance the traceability of the project over time, which is a significant advantage for both developers and stakeholders. This can facilitate software maintenance and further development.

> "*Developer 1: At this point, even with the small number of things that are annotated and the lack of efficient querying, I still think that it has value or will have value down the line if we continue in this way. If we had this type of RM in place when starting a security process where we go through documents with specific requirements where we would then annotate how we meet these requirements, we could then two years later use a command to show - what was our reasoning for us meeting this requirement from this company?*
>
> *In the same vein, if we have a configuration that we do not understand the meaning of later on, we could write a small text indicating what this part of the config pertains to and why this is switched on in this service.*

> *[...] If the work we have done these weeks had been started years ago, I think it would have brought huge value.*
>
> *Developer 2: I also think that, when looking at other stakeholders, if we had this process from the start and explained it to them, there wouldn't be any resistance to its inclusion.*"

The role of requirement management in agile development: Despite initial difficulties, RM can strengthen agile processes by improving the clarity and precision of requirements, potentially leading to higher quality and better maintainability in the final product.

> "*... compared to user stories which are more time-bound and a way to visualize the workload for this sprint, if you create good requirements and prevent them from becoming another way of tracking user stories, they could become more timeless in comparison. They would then explain the list of features within the system and requirements for these features which would be another way to track these instead of looking into the worklog.*"

RQ2: Factors influencing developer stakeholder acceptance or resistance

Acceptance:
Communication and collaboration on requirements: Regular and clear communication with stakeholders about the benefits and process of RM can increase acceptance.

Adaptation and flexibility in requirement management tools: Tools that can be tailored to the specific needs of the project and stakeholders, seamlessly integrated into existing workflows and/or includes features that are valuable across all projects are more likely to be accepted.

> "*And there's another perspective: Let's say that we have very detailed existing traceability or used the requirements model from the start. If the tool in question would be more user-friendly such as having a GUI, generated HTML reports that connect somewhere to the point where you could in theory look at what requirements we fulfill when somebody asks if a platform includes a specific requirement. If the model is available on a sufficiently high level there would maybe exist a requirement for a component that states that we fulfill that requirement along with its definition. That type of functionality and accessibility would be superb, especially for those who are not as familiar with the project compared to us developers.*"

Resistance:
Meta-model structure and efficiency: If the chosen meta-model does not adequately reflect the project's and stakeholders' needs, it may lead to resistance to RM integration.

Technical and organizational challenges: If the RM tool is perceived as too cumbersome or time-consuming, it may encounter resistance from both developers and stakeholders.

> "*It's hard to invest in a tool that is this niche in its scope and acceptance. [...] For an industry where fulfilling requirements is vital, such as those concerning space travel or flights, bookkeeping becomes very important and it is very reasonable for a project to have more lines of documentation than code. However, in our case, we would maybe like to add some random JavaScript and then just quickly see how that looks, and thus the documentation would feel onerous. [...] Another point is how this would look over time. Would the requirements be maintained or simply be left in the history, will more links or wrappers be added, how will the readability of the code be affected?* "

RQ3: Adaptation of agile processes to support RM

The role of requirement management in agile development: agile processes can be adapted to include RM by integrating requirement management activities into sprint planning and retrospectives, where developers can reflect on the quality of requirements and how they have been fulfilled during the sprint.

Historical and future traceability of requirements: Developing tools and processes that facilitate easy updating and tracking of requirements in conjunction with code changes can help adapt agile processes to RM.

> "*We work with a team and a living process. The team changes, the stakeholders change, we gain more experience and how we work as a whole changes over time. New people, new colleagues arrive and we have a changing process but the model we use is most times immutable. In some instances, you would like to have this solid model since there can be incredible pressure on delivering quality and maintaining a standard. But I also think that the standardization of things can many times get in the way of how the team would naturally evolve since you would always adhere to its very strict nature, so having an artefact that can live and change WITH the team would maybe be preferable.*"

Adaptation and flexibility in requirement management tools: By using RM tools as part of an agile process, developers can become more disciplined in documenting and linking requirements directly to code, thereby contributing to a more comprehensive and accurate requirement management process.

> "*I thought of the scenario where you have several T-Reqs-elements that do a bunch of different things and then another developer runs* `treqs list` *two months later to check how I fulfilled a certain feature. If I had wrapped that feature requirement into a function that does about ten different things, the other developer would see my bad code and wouldn't quite know what my T-Reqs annotation means. So, to make this RM tool work I had to clarify the code structure so it fits the annotation. [...] So I at least thought that, when I did coding work within the context of this sprint, it was nice to have in that way.*"

4.1.2.2 Metrics

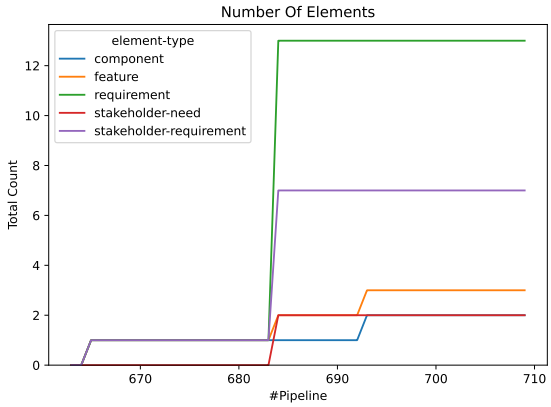
As per Sections 4.1.1 and 3.5.2, metrics were collected during each execution of the pipeline and are depicted in Figure 4.4. Initially, there was a notable quiet period at the start of the sprint, followed by a gradual introduction of *stakeholder-requirements*. However, the number of *stakeholder-needs* remained relatively low, reflecting the team's efforts to differentiate between needs and requirements as perceived by stakeholders.

In the later stages of the sprint, while the count of higher-level elements remained consistent, links to these elements began to emerge alongside the implementation of concrete *features*. As more features were integrated, the interconnectedness of the requirement model increased, evident from the steady rise in in-links to *components* and *features*. Concurrently, as implementation work commenced within the sprint, *features* started to establish connections with higher-level elements in the model.
It's worth noting that the sudden spikes observed across all charts were primarily attributed to a bulk of changes stored in a separate branch, which was merged at that specific point.

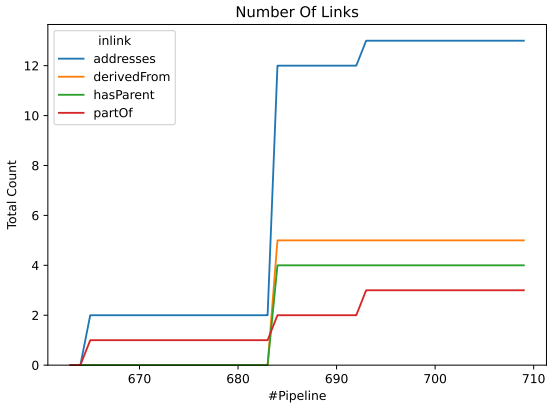
4.1.2.3 Conclusions

The results gathered from ICI steered the direction of our new implementations and improvements for ICII while also gathering insightful knowledge that will be used to answer our research questions.

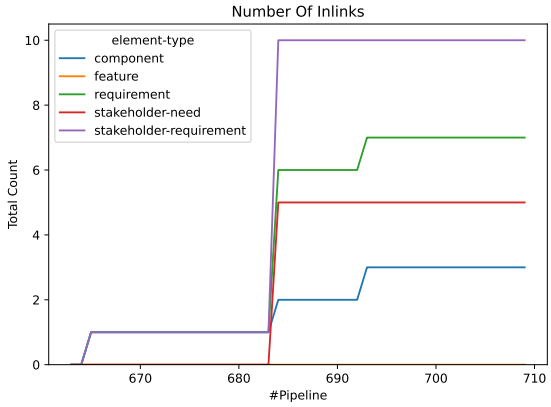
While the main purpose of the metrics for the first iteration is to validate the model chosen for the project, we can observe smaller additions to the metrics that can show proof of how the team divided their high-level and low-level elements into different parts of their development process. We can see in Figure 4.4a that all the *stakeholder-need*, *stakeholder-requirement* and *requirement* types are only added from a single pipeline run and there are no additions for the rest of the sprint cycle. However, we do observe an increase in both the number of *component* and *feature* types in the model. These types are added as developers transition from the planning-focused cooldown stage to executing their sprint items, aligning with our observations.



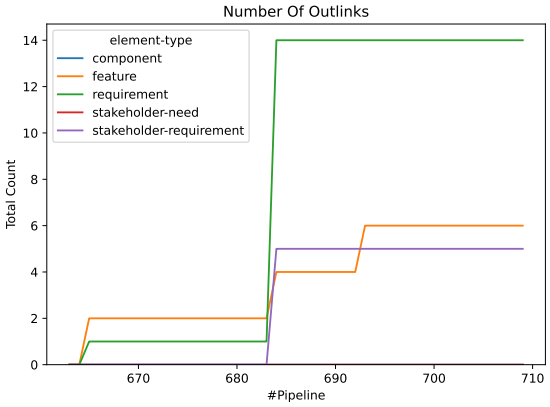
(a) Number of elements



(b) Number of links in the model



(c) Number of in-links per element type



(d) Number of out-links per elment type

Figure 4.4: Plots displaying four different metrics collected throughout ICI. The x-axis represents the '#Pipeline' scale, which corresponds to the ID of each pipeline run. The ID increments by 1 with each execution. The y-axis indicates the values of the respective metrics being measured.

The findings extracted from the reflection meeting also laid much of the foundation for the discussion present in Chapter 5 as the balance between the added benefits of RM and its intrusiveness on development is the main issue that determines its acceptance.

The following three weeks of our study were dedicated to refining the final round of our field experiment where the additions consisted of improvements to our tool of choice and changes in the meta-model to further facilitate better results.

4.2 Second Intervention Cycle

The design of ICII is based on the results and observations obtained from ICI. Unlike the previous cycle, the team is now more familiar with the concepts of RM and the tool used. With a baseline understanding established, we feel more confident in further integrating RM and T-Reqs into the workflow and day-to-day activities of the team.

4.2.1 Preparatory Phase

In anticipation of ICII, and based on the feedback on the previously altered variables, we tweaked and refined the existing intervention points. The preparatory phase culminated in the following adjustments and additions:

Blocking model consistency checks in Pipeline

As the team is becoming more comfortable with adopting RM, the members expressed not only their desire but need to have the model consistency checks blocking in the pipeline. Blocking in this context means that if any of the elements in the model does not adhere to the TTIM, the pipeline run will stop at the RE stage and will therefore hinder a Pull Request or merge to the main branch. This helps ensure the overall consistency of the model. This is a proactive measure to maintain the requirement compliance, quality and reliability of the software being developed.
The following line has been updated to issue an blocking error in the pipeline in case `$ treqs check` fails:

```
echo "##vso[task.logissue␣type=error]The␣model␣is␣
    inconsistent."
```

Refine requirement meta-model

To better capture the domain of the software being developed and maintained by the team, and by an iterative discussion process, the meta-model (see Figure 4.2) used in ICI has been updated to align with the nature of the artifacts being modelled by the team. We also wanted the changes to be easy to migrate to alleviate friction. The team itself has been very proactive in the refinement process, which reflects their sense of ownership of the meta-model.

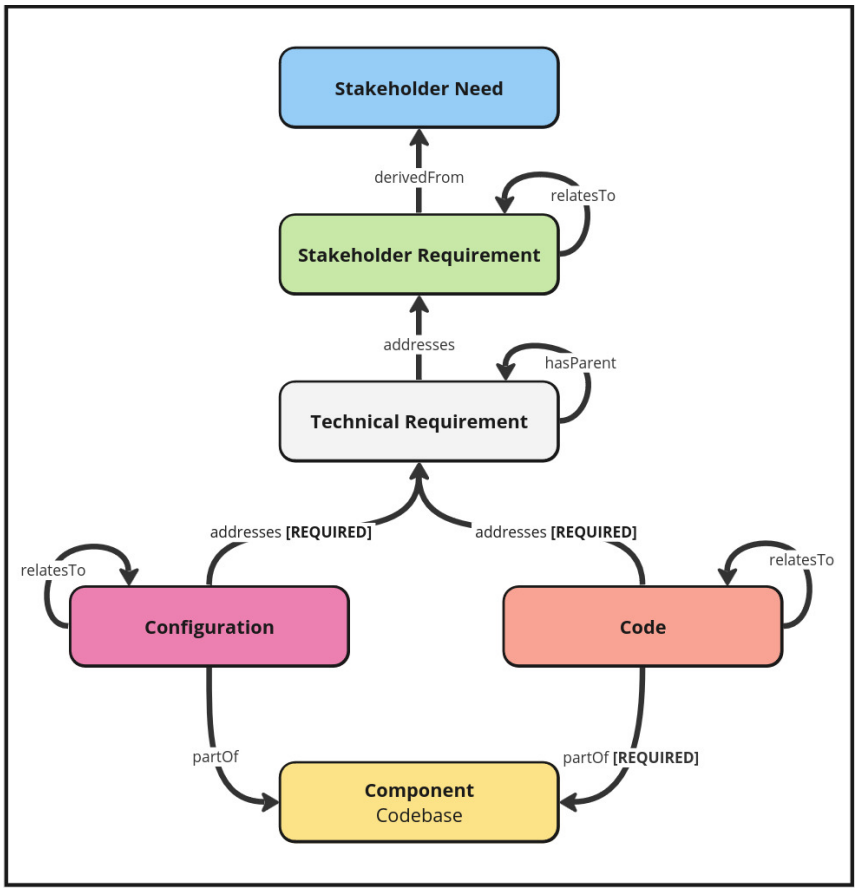
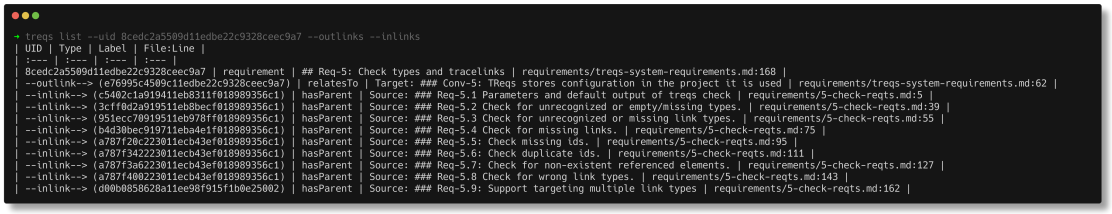


Figure 4.5: Type and Trace Information Model for the second intervention cycle, featuring 6 distinct types, 3 mandatory traces, and 3 reflexive traces.

For the most part, the updated meta-model (see Figure 4.5) retains the same elements as the previous version. However, we have identified the following differences:

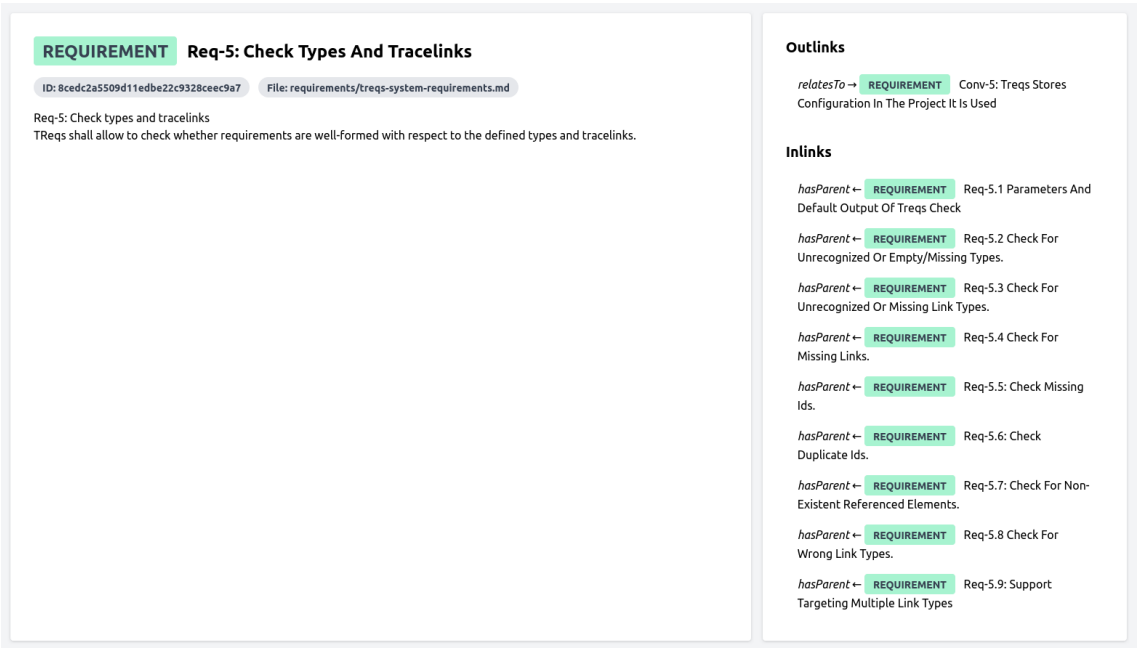
- Renaming of *requirement* to *technical-requirement*, to avoid ambiguity when discussing the requirement model and to indicate that the team is the responsible party for defining this type of requirements.
- Clear hierarchy between *stakeholder-need* and *stakeholder-requirement*. Additionally a *technical-requirement* cannot directly link to the top-level type *stakeholder-need*, but indirectly via a *stakeholder-requirement*.
- Splitting the type *feature* into *code* and *configuration*. This adds granularity to the requirement meta-model as a whole, as the team not only maintains and develops code, but also configuration files. Both new types share the same link types as *feature*, but now a *configuration* must not be a *partOf* a *component* as configurations can be for third-party components not owned by the team.

A YAML representation of the requirement domain meta-model is found in Appendix E.



```
→ treqs list --uid 8cedc2a5509d11edbe22c9328ceec9a7 --outlinks --inlinks
| UID | Type | Label | File:Line |
| :--- | :--- | :--- | :--- |
| 8cedc2a5509d11edbe22c9328ceec9a7 | requirement | ## Req-5: Check types and tracelinks | requirements/treqs-system-requirements.md:168 |
| --outlink--> (e76995c4509c11edbe22c9328ceec9a7) | relatesTo | Target: ### Conv-5: TReqs stores configuration in the project it is used | requirements/treqs-system-requirements.md:62 |
| --inlink--> (c5402c1a919411eb8311f018989356c1) | hasParent | Source: ### Req-5.1 Parameters and default output of treqs check | requirements/5-check-reqts.md:5 |
| --inlink--> (3cff8d2a919511eb8becf018989356c1) | hasParent | Source: ### Req-5.2 Check for unrecognized or empty/missing types. | requirements/5-check-reqts.md:39 |
| --inlink--> (951ecc70919511eb978ff018989356c1) | hasParent | Source: ### Req-5.3 Check for unrecognized or missing link types. | requirements/5-check-reqts.md:55 |
| --inlink--> (b4d30bec919711eba4e1f018989356c1) | hasParent | Source: ### Req-5.4 Check for missing links. | requirements/5-check-reqts.md:75 |
| --inlink--> (a787f20c223011ecb43ef018989356c1) | hasParent | Source: ### Req-5.5: Check missing ids. | requirements/5-check-reqts.md:95 |
| --inlink--> (a787f342223011ecb43ef018989356c1) | hasParent | Source: ### Req-5.6: Check duplicate ids. | requirements/5-check-reqts.md:111 |
| --inlink--> (a787f3a6223011ecb43ef018989356c1) | hasParent | Source: ### Req-5.7: Check for non-existent referenced elements. | requirements/5-check-reqts.md:127 |
| --inlink--> (a787f400223011ecb43ef018989356c1) | hasParent | Source: ### Req-5.8 Check for wrong link types. | requirements/5-check-reqts.md:143 |
| --inlink--> (d00b085b628a11ee98f915f1b0e25002) | hasParent | Source: ### Req-5.9: Support targeting multiple link types | requirements/5-check-reqts.md:162 |
```

(a) Traditional approach to listing requirement information



(b) Generated report page for a requirement

Figure 4.6: Comparison of viewing the same requirement using the traditional *$ treqs list* (top) and the experimental report generation feature (bottom) introduced in ICII.

Generating requirement model report

To enhance the visibility of the requirement model, an experimental feature has been introduced to generate a catalogue report reflecting the current state of the model. This serves as a more user-friendly approach for tracing, exploring, and communicating the model within the team and to external stakeholders. Unlike the previous method, which involved navigating through multiple `$ treqs list` commands outputting markdown tables with limited information, the customized reports offer several advantages: The default report template facilitates fuzzy search for requirements and enables filtering by type, streamlining the exploration process. Additionally, it provides a brief overview of outgoing and ongoing links, incorporating various customized features leveraging web technologies, in contrast to terminal-based outputs. Overall, this initiative aims to improve accessibility and usability, making the requirement model more comprehensible and actionable for all involved parties, see Figure 4.6 for comparison.

Automated report generation in DevOps pipeline

Building upon report generation, the feature was integrated into the team's Azure DevOps pipeline to produce a self-contained, single-page report detailing the current state of the repository. Due to limitations of the Azure DevOps platform, the report had to be in Markdown format, necessitating the use of a custom-tailored Markdown template outlined in Appendix F. The report begins with a table containing different elements: labels and types. Each row in the table is linked to the element's section in the subsequent part of the report, which provides additional information such as outbound and inbound links, all of which are clickable and direct users to the relevant element. Moreover, the report enables navigation to the precise version of the element used for generating the report. The technical details on the integration and generation of the report in the pipeline are outlined in Appendix G.

Tracking requirement changes

To facilitate the agile process within the team and leverage the inherent nature of version-controlled requirement modeling, a helper script was introduced. This script lists the new requirements introduced since a specific commit. The primary purpose is to compile a comprehensive list of all additions to the requirement model by the end of a sprint. Additionally, it can be utilized to display requirement changes between branches or since the last commit. This approach not only facilitates agile processes but also provides a clear overview of project scope evolution, aiding in sprint planning and decision-making. This feature was implemented as a helper Bash script and not into T-Reqs code-base for ease of integration and future modification by the team. It has a minimalist interface where `$ ./treqs diff HEAD~2` shows the added requirements since the last two commits. The implementation of the feature is shown in Appendix H.

Intervention Categorization

In Table 4.1, we categorize the changes for ICII based on their relevance to different aspects of tooling and processes. The features are classified into three main categories: RM Tooling, DevOps, and agile Process. Each feature is assigned to one or more categories to highlight its role in supporting various aspects of project management and development. This categorization helps in understanding how different tools and processes are anchored to the overall intervention flow and research questions.

Feature	RM	DevOps	Process
Tracking requirement changes	✓		✓
Automated report generation in DevOps pipeline		✓	
Generating requirement model report	✓		
Refine requirement meta-model	✓		✓
Blocking model consistency checks in Pipeline		✓	

Table 4.1: Categorization of interventions introduced in ICII

Observations

The observations gathered from the second and last intervention cycle were focused on refinement and experimenting with the new features and model changes that the team now had at their disposal. As the team had become accustomed to the changes our interventions had made there were minimal issues in using it throughout the cycle. Instead, time was spent on examining features that enhanced the final traceability visuals and querying capabilities. Each team member also had a picture of Figure 4.5 at their desk, which could potentially show their interest in RE/RM activities and their willingness to make them work based on their results. More nuanced discussions about the model occurred regularly as it was now easier to understand the different elements and its hierarchy, including discourse about the potential advantages of a simpler model with fewer vertical levels but a free level of granularity within the lower level (elements below *technical-requirement*. Another point of focus was in the form of looking at tests as requirements, e.g. if the stakeholders want a specific feature, a test would be present in the model to represent it and its pass/fail result would either fulfill or break it.

The exploration of more visualization options came in the form of using tools such as Gephi and Neo4j to produce node networks of the system when exporting the elements of T-Reqs that also inhabit more powerful querying. One of the visuals produced from this approach is used as the cover image of this paper and the initial impression of these new options was that they are better suited toward showing external stakeholders the totality of the system or how a new need has been implemented after a sprint. The interventions for ICII focused on report generation both in HTML form and within the pipeline were received well. However, the team felt that they were more geared toward benefiting developers, thus the exploring of other options was made to widen the range of use cases for this aspect of RM. Lastly, the team agreed that the ability to track requirements through `treqs diff` would be valuable when working within sprints, but the feature was not used to its full extent as the work done in the ICII sprint was rather sparse due to factors beyond the team's control.

4.2.2 Evaluation of ICII

After the second intervention cycle and the field experiment, the evaluation focused not only on collecting feedback about the second intervention but also on assessing the study as a whole. While considerations for future improvements and changes were present, our primary focus was on addressing the research questions.

To enhance the evaluation of the results from this cycle and the overall study, we presented our significant findings to a panel of industry experts in the field of RE to gain their insights and consult their unbiased and neutral opinions on the study's findings up to that point. These discussions provided us with a valuable perspective for evaluating this intervention cycle.

Reflection Meeting

The reflection meeting for ICII was executed in the same vein as per Section 4.1.2.1 apart from changing the questions stated and executing the emergent coding manually without tools. The questions for this cycle can be found in Appendix I and focus more on gathering insight for our final evaluation and finding potential focus areas for future work outside of the scope of this study. Described below are the themes extracted from the meeting, categorized within our research questions. Compared to the ICI, some of these themes are more related to the specific tools used in the study due to the focus of our questions:

RQ1: Challenges and Benefits of Introducing Requirement Management in DevOps for Software Developers

Challenges:
DevOps pipeline error needs to be more concise and less noisy: To properly fit within a DevOps pipeline and especially within the current intervention where a fault in the model results in a blocking error, the errors that occur must be concise and easily found to not become cumbersome for developers.

> "*I didn't know which part of the pipeline I was supposed to look at. I accessed one of them and was greeted by 6000 [log messages], I scrolled down for a while and then decided - Alright, I'll let my colleague handle this one.*"

Difficult to show the value of RM at the start of a project: When initially approaching or presenting RM into an agile project, the time-cost and ambiguity in its implementation make it an unattractive option, especially as it lacks in showing tangible results early.

> "*Developer 1: I think that it is meant to be present for a while until you open up the option where its possible to go back and look extensively at the project history.*
>
> *Developer 2: Yes, I think this also comes with experience over time, where a senior developer understands how this is valuable because they have been through so many projects throughout their career.* ***The challenge is in making people understand the value from the start.***
>
> *Developer 3: I listened to a podcast where they discussed the factor of where you're practically forced to write down the requirements of the customer as early as possible in order for the customer to learn how to formulate 'good' requirements further down the line.*"

Advantages:
Streamlined multi-level tracing leads to better system design and further understanding of new requirements: When a developer and customer can understand the com-

plete timeline and the sources that a feature comes from, it forms an understanding of both the communication channels between the two parties and more concretely answers what the system on a broad level is capable of.

> "*I think it once again comes down to the 'how' of it all. How do you use the requirements model? For us developers, the value may be there when we can look upward and find out - why are we doing this? It's more extensive in that way since we don't just get the fragments of requirements. It helps one to understand the totality, which can then help with creating a technical solution that better addresses what the customer needs. And if we understand what they really want better we can start to say no and present a solution that is better from a technical standpoint.*"

The advantages of blocking pipeline stage outweighs the downsides: Since the model itself decides on the strictness of the elements and the links in between them, the pipeline stage that ensures that the project adheres to these rules should always be blocking to be sure that developers adhere to their set plan.

> "*Q: The other option would be to change to a non-blocking requirements check in the pipeline to where your PR's merge without paying attention to the model being correct or not. Would you prefer it that way?*
>
> *If it would work that way, we would rather continue to work with it being a blocking stage. In the other case, if you were short on time then you would just skip that step and gradually build up a debt in the model and system.*"

RQ2: Factors influencing developer stakeholder acceptance or resistance

Acceptance:
Make it possible to adjust the granularity of low-level elements: When using RE/RM to benefit other stakeholders, the place of focus is on tracing between levels or describing the high-level elements of the system. While the high-level abstraction should adhere more to a simple model, the low-level requirements can be made more granular through additional elements for developers to map their system in their own optimal way.

> "*With the new meta-model, I think it 'catches' the system better. When I put down the annotations, it was code that I put on code that I wrote. Comparing this to just having 'feature', it felt semantically better. It felt strange when labeling a configuration as a feature before since it was simply that, something that feels too 'cheap' to be a fully-fledged feature.*"

Better forms of visualization quickly show value: When visualization implementations are made that separate the result from the CLI, you will find it much easier

to both present your work for other stakeholders but also keep track of your own through every sprint much quicker.

> *We need to do further work on the HTML-report because I think both it and the Markdown format report are very important. It would be even better if we had a smart meta-model that could generate the reports to be more effective.*

Resistance:

RM tooling need to be context-aware or integrated into various Doc-tools: Especially for code and configurations that do not require large amounts of lines, the XML-inlining and obtrusive look that T-Reqs inhabits makes it a sore spot for developers since you trade code readability for code traceability.

> "*To run and embed XML still feels awful. We see that we were more diligent with tools like JS, Java and Kotlin Doc in those specific files. These tools should be integrated with T-Reqs so that you can create your annotations that are then understood and translated into a documentation system. Instead of having XML everywhere in favor of context-aware tooling, these would both eliminate awkward parts in the T-Reqs Report along with in the source files themselves. For example, your requirements seamlessly end up in your JS Doc and your T-Reqs Report will add it based on the JS comment made to annotate the element.*"

Small teams no matter the distance or complexity will see the time-cost of RM as even harsher: One week for each developer is very valuable time for all stakeholders involved. As teams grow in size, it is much easier to assign time for administrative and management duties to a single person to save time for everyone. However, when this overhead is found within small teams, the time that effective communication channels or a small distance between members saves still does not alleviate the additional time that RM brings into the workflow.

> "*We have a very high level of complexity since we work with a lot of different system components and a lot of small parts within those components. [...] If RM could help with these complexity problems, it would be great. And I would go on to say that with small teams with a close distance to the customer like us, [RM] would still be valuable, but in these teams, the admin and management parts cost so much more if you are only three people. If we would be e.g. eight people, you could put one person to work on administrative things half of the day, but this is not achievable here.*"

RQ3: Adaptation of agile processes to support RM

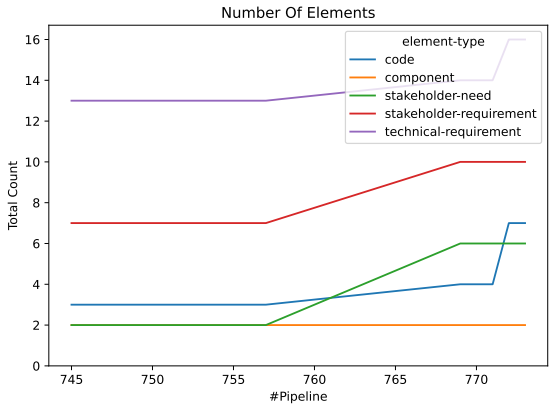
RM in agile projects facilitates better awareness for developers and other stakeholders: Even when work items are created for a sprint in a storyboard, the actual work that comes out of these may be much different compared to what was planned simply due to the way agile works and promotes constant change. RM could show a better picture of this work and would assist in the dialog between parties when figuring out what is actually being worked on and created in a project.

> "*Developer 1 If we had [RM] in place before and had used it for a while, then I think the complicated discussions we have had with the stakeholders could have gone a different way. Let us say they ask outright - What are you doing here, right now? - We could just run treqs diff and show sprint-by-sprint what we did!*"

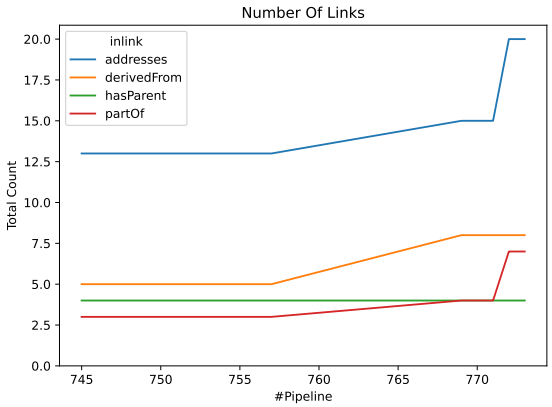
> "*Developer 2: What we had on the sprint board compared to how it actually looks since what is written initially rarely holds since we encounter new things that change the plan.*"

RE/RM can itself be a way to avoid double bookkeeping: In many agile projects, the taskboard exists to both plan for the work ahead and to document the work that was done before. However, these largely exist in a separate space and may look vastly different when looking at the end result, a problem that RE and RM may be able to solve.

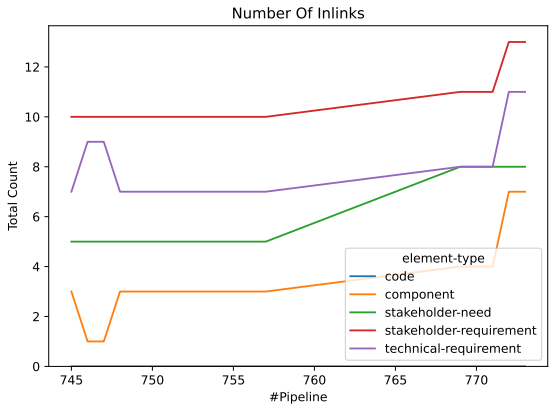
> "*We have this confusing situation where our taskboard acts as both a way to organize us daily, but also in some ways to show our documentation and requirements. If you could formalize a requirements management system, this could become an all-arounder tool in the same vein which I would prefer. In that case, the source tree would not have a separate filesharer, external taskboard or issue tracker. What remains is still how to separate a user story from a requirement when handling these things, this distinction needs to be defined clearly.* "



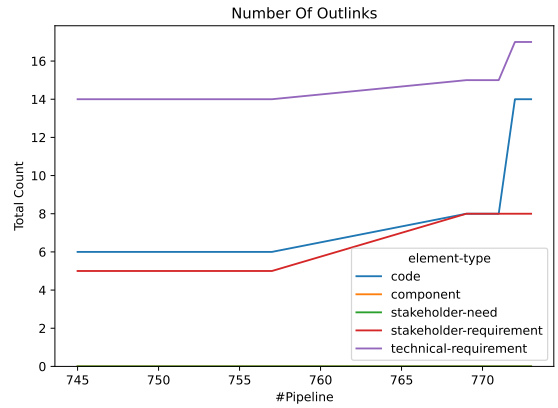
(a) Number of elements



(b) Number of links in the model



(c) Number of in-links per element type



(d) Number of out-links per element type

Figure 4.7: The plots display four different metrics collected throughout ICII. The '#Pipeline' scale represents the ID of each pipeline run, with each execution incrementing the ID by 1. The symmetrical uptick and downtick in the number of in-links to elements of types *component* and *technical-requirement* can be attributed to the gradual migration from the old TTIM to the new one.

Metrics

Similar to the previous intervention cycle, we collected metrics (see Figure 4.7) from each pipeline run throughout the second intervention cycle. We observed a quiet period at the beginning corresponding to the cooldown week, during which few elements were added to the model. It's noteworthy that no new elements of type *component* were added during this time, which is not unusual, as the number of components in a micro-service architecture tends to remain stable unless a stakeholder need necessitates adding another. Halfway through the sprint, we observed a sharp increase in higher-level abstraction elements such as *stakeholder-need* and *stakeholder-requirement*. Towards the end of the sprint, there was a notable increase in elements representing actual implementations i.e. elements of type *code* (see Figure 4.7a). Most links (both inward and outward) were added in the final phase of the sprint, indicating that the team connected elements as code implementation progressed, suggesting a bottom-up linking process (see Figures 4.7b, 4.7c and 4.7d).

4.3 Final Results

In this section, we present the final results and address the research questions of the study. Each research question is linked to a an aspect similar linking it to the actions introduced in the intervention cycles to facilitate the discussion. To offer actionable answers, we provide a summary of the response to each research question before delving into deeper discussion.

4.3.1 RM and Tooling: Addressing Challenges and Leveraging Benefits

RQ1 What are the challenges and benefits of introducing requirements management as it pertains to software developers when integrating it into a pre-established DevOps workflow?

Introducing requirements management into a pre-established DevOps workflow can bring both challenges and benefits for software developers, we identify the following challenges:

- **A** RM Tools are not as extensive; incomplete feature set.
- **B** RM Lacks instant gratification; requires upfront investment.
- **C** Agile RM lacks a concrete standardization topic for developers.
- **D** Requires special know-how to integrate into DevOps.
- **E** Top-down or bottom-up modeling — which one to use and when and by whom?
- **F** RM Error messages in the pipeline need to be concise and less noisy.

We also identify the following benefits:

- **G** Transparent communication between developers/clients.
- **H** Ability to trace low-level requirements to high-level requirements helps with better overall requirements and system design, system maintenance and further development.
- **I** Improved querying capabilities of the requirements, allowing for knowledge extraction.
- **J** Strong and enduring RM processes facilitate discussions in the consulting industry.
- **K** RM helps with onboarding, explaining and understanding the system.

In adopting agile RM, developers encounter several challenges alongside notable benefits. One significant challenge is the limited toolset available, often lacking features critical for comprehensive adoption. While the basic functionality of bookkeeping requirements is present in most tools, this only covers the initial phase of working with requirements. For an effective use of the time and effort investment in RM-tools, more knowledge extraction, quality-of-life and simpler interface makes it more approachable initially and makes it more appealing to a larger section of users. In the context of RM tools such as T-Reqs, which embed annotations within code and other text-based artifacts, the demand for custom syntax highlighting,

template snippets, and auto-completion becomes more apparent but is currently lacking. Integrating support for these features through editor extensions would not only enhance adoption but also mitigate the risk of typos and other inconsistencies in the annotations. This deficiency necessitates a careful evaluation of available tools and future tailored development to ensure they align with project needs and goals.

Another challenge is the lack of instant gratification; successful implementation requires a significant upfront investment of time and resources. This initial investment can be difficult to motivate because it relies on the future expectation that RM will yield dividends. While both the team and experts acknowledge that the benefits of RM are delayed, experts anticipate greater value after RM efforts have been in place for an extended period and not just in the matter of a couple sprints.

Additionally, integrating RM into DevOps workflows demands specialized expertise, which can be challenging for teams unfamiliar with this integration. The technical intricacies affect the depth and extent of integration with RM tools and the cohesion with other agile process artifacts and activities such as backlog and sprint boards. The choice of DevOps platform and requirements tooling significantly impacts the integration's effectiveness. While the ability to publish requirements reports directly from the DevOps platform is supported by the tooling, it remains contingent on the team's platform choice and necessitates technical insight and expertise into the platform's specifics as a whole. Moreover, managing error messages in the pipeline proves challenging, as concise, less disruptive notifications are essential for efficient workflow, while at the same time providing more contextual logs proves essential in locating and fixing any inconsistencies.

Choosing between top-down or bottom-up modeling further complicates matters, raising questions about which approach to employ and when, and which team members should lead these efforts. As developers tend to appreciate working with bottom-up modelling as they tend to interact more with the concrete artefacts in form of code, configuration, testing or documentation. On the other hand, project managers and external shareholders focus more on the abstract needs, demands and non-functional requirements in their verification and validation efforts.

Despite these challenges, agile RM offers significant benefits. Its flexible modeling language allows for precise capturing of the domain, facilitating transparent communication of the development process state among developers and clients. This enhances project understanding and collaboration.

Additionally, the capability to trace requirements from low to high levels of abstraction supports robust system design, maintenance, and future development, providing a shared and agreed-upon ledger of the process and product. This helps mitigate potential conflicts among stakeholders in the future. In that regards, the key feature of querying capabilities within requirements tools facilitate knowledge extraction, contributing to more informed decision-making and system evolution and conflict

resolution, especially within the consulting industry, fostering effective project management and client engagement.

Ultimately, regardless of the project scope or industry specifics, the adoption of RM facilitates onboarding, system comprehension, and explanation, thereby reinforcing its value in driving project success and sustainability.

4.3.2 RM and Developer Experience: Factors Influencing Acceptance and Resistance

RQ2 What factors influence developer acceptance or resistance to incorporating requirements management into a DevOps framework?

We identify the following factors that affect the developers' acceptance and resistance to RM tools:

- **A** Enhance RM tool adoption by allowing extensions and integrations.
- **B** Developer-centric tooling: Minimize the gap between implementation and modeling.
- **C** Effective management of transient files in git.
- **D** More granularity for low-abstraction elements in the model.
- **E** Fail often, fail early.
- **F** Improve knowledge extraction.
- **G** Idiomatic and context-aware inlining of requirements with textual artefacts.
- **H** Higher cohesion between requirements and tests.
- **I** Effectively leverage flexible RM modelling tools by making conscious decisions.

Various factors influence the acceptance and resistance towards adopting RM tools and methodologies. While the source of the factors derived from the experiment is limited to developers, the results still capture factors that involve or benefit other stakeholders since the communication and formulation of requirements is a crucial part of development that developers want to improve upon.
Emphasizing developer-centric tooling underscores the importance of bridging the gap between implementing a feature and modeling it, fostering agility and efficiency in requirements processes. The choice of the open-source tool T-Reqs as the RM tool for this study was successful because it allowed for the extension and implementation of developers' requests for tools on demand, increasing their sense of ownership over requirements and the associated tooling. Lowering the barrier for tool expansion by incorporating features like templates and historical diffs plays a crucial role in easing adoption, reducing entry barriers, and enhancing usability.

Conversely, and similar to the discussion of *RQ1.E* above, challenges arise when determining the suitable approach between top-down or bottom-up modeling, posing questions regarding the optimal strategy based on project scope, team dynamics, and stakeholder needs. However, the acceptance of a flexible modeling language proves

invaluable, accommodating diverse requirements and evolving project demands with enhanced adaptability and scalability.

Managing transient files within repositories can pose challenges, particularly due to varying developer workflows. Emphasizing streamlined version control practices is essential to maintain project integrity and collaboration. The development process often generates side artifacts that are necessary for developers to perform their tasks but shouldn't be considered within the RM process. Additionally, similar to *RQ1.F*, refining error messages in the pipeline is imperative to minimize distractions during requirements analysis and validation. This refinement helps reduce the time between feature implementation and merging into the master codebase, ensuring efficient and effective communication throughout the development cycle.

From the developers' perspective, advocating for more granularity in modeling low-abstraction elements empowers detailed and precise representation of requirements, facilitating comprehensive understanding and analysis on the developers' part. Embracing a *fail often, fail early* mentality encourages developers to embrace rigorous requirements management checks, cultivating a culture of continuous improvement and quality assurance.

Moreover, leveraging improved querying capabilities enhances the extraction of valuable insights from requirements data, empowering developers with actionable knowledge for informed decision-making and a shared starting point for discussions revolving around the development of new features and maintenance of existing ones.

When embedding requirements with static development artifacts like code, configuration, and documentation, it's crucial to adopt an idiomatic and context-aware approach to preserve readability and maintain the integrity of both the codebase and requirements documentation. Inlined annotations should be used thoughtfully, considering that they could disrupt the flow of textual artifacts, especially when the annotated feature is dispersed across multiple files or locations within the same file. This requires careful consideration to ensure that the annotations enhance understanding without impeding the natural readability and organization of the associated artifacts.

Furthermore, enhancing cohesion between requirements and tests fosters comprehensive coverage and validation of project specifications, ensuring alignment between development goals and business objectives.

Lastly, evaluating the flexibility of tools like T-Reqs requires deliberate decision-making regarding its application and utilization, emphasizing dynamic modeling choices to maximize its benefits effectively within the project context in the long run.

4.3.3 RM within Agile Processes: Addressing Challenges and Implementing Effective Actions

RQ3 How can the existing agile processes be adapted to support RM efforts?

Adapting existing agile processes to effectively support RM efforts involves addressing several key challenges:

- **A** Agile RE/RM lacks a concrete standardization topic for developers.
- **B** Balancing strictness of the model based on the specific use case of the project: simple, flexible, complex, rigid (granularity vs. ambiguity).
- **C** Define the overlap or lack thereof between user stories and requirements.
- **D** How to integrate RM efforts with compliance efforts.

Actions that need to be implemented to make agile processes more efficient from an RM perspective:

- **E** In a sprint, model higher-level abstractions first, then move on to mid-level elements, and finally low-level elements.
- **F** Allow for revision of elements to be done anytime during a sprint.
- **G** Define clear RM responsibilities for team members for high-level abstractions; the product owner is the caretaker of high-abstraction elements.
- **H** Define clear RM responsibilities for team members for mid to low-level abstractions.

Adapting existing agile processes to effectively support RM efforts involves addressing several key challenges and implementing specific actions.

One challenge is the lack of standardized practices within agile for RM, particularly in providing clear guidelines for developers. Another challenge lies in balancing the level of detail in models based on project needs—whether it should be simple yet flexible or complex but rigid, navigating the trade-off between granularity and ambiguity, the answer to which depends on the project characteristics and the maturity of the surrounding RM tooling and its integration with the DevOps processes of the organization.

To tackle these challenges, a series of actions can be implemented within agile sprints. Firstly, during a sprint, begin by modeling higher-level abstractions, progressing to mid-level elements, and then low-level details. This phased approach ensures a systematic development of requirements within the sprint timeframe. Additionally, allow for revising elements at any point during the sprint, enabling flexibility and adaptation as insights evolve.

When transitioning to new models, undertake migration in incremental steps to minimize disruption and ensure smoother adoption both mentally and syntactically.

Another challenge involves defining the relationship between user stories and requirements and integrating RM efforts with compliance needs. To address this, establish clear responsibilities within the team. Assign the product owner as the custodian of high-level abstraction elements, while other team members are responsible for mid to low-level abstractions. This division of responsibilities ensures clarity and accountability within the agile framework.

In summary, adapting agile processes to support RM efforts involves addressing challenges through specific actions, such as phased modeling within sprints, flexible revision practices, incremental model migration, and clear delineation of responsibilities for different abstraction levels within the team. By implementing these strategies, agile teams can enhance their capability to manage requirements effectively while maintaining the agility and adaptability characteristic of agile methodologies.

4.3.4 Summary

RM can be costly, especially for smaller teams. However, RM skills are developed and refined over time. Early adoption of RM also benefits clients by ensuring clearer project goals and expectations.
The true value of RM is evident in three key aspects:

- **Internal Distance**: This refers to the distance or separation between team members, considering factors such as the number of teams involved in the project and the specialization of team members. The greater the internal distance, the more beneficial RM becomes in maintaining alignment and communication within the project.
- **External Distance**: This involves the distance between the project team and the validation loop with external stakeholders or customers. It can be likened to the length of the feedback loop—where longer loops necessitate stronger RM practices to ensure that project outcomes align with external expectations and needs.
- **Task Complexity**: RM is particularly valuable for projects characterized by high complexity or extended timeframes. The duration and age of the project contribute to its complexity, making effective RM crucial for managing evolving requirements and expectations over time.

In summary, while RM may present initial challenges and costs, its long-term benefits in improving internal and external alignment, especially in complex projects or those with extended durations, justify its adoption and development within agile practices.

5
Discussion

This chapter will discuss and evaluate the results and findings from our study while also examining potential solutions or future work based on the research questions and problems presented in this paper. Lastly, this chapter will also discuss the limitations and delimitations of our study as well as the potential threats to validity.

The solutions and answers within this section will be picked from the results based on their generalizability within the RE and agile space. While the in-depth observations are based on the use of our specific tool of choice, much of the findings from this study can easily be viewed as a situation where the software or RE-specific activity chosen is not a critical factor.

5.1 Interpretations of Findings

> "*Right now, there is no carrot nor stick! We would need more of both parts.*"

The discussion of our final results in this study can be summarized through several key themes: *intrusiveness, obtrusiveness, clarity, flexibility, tool maturity and project characteristics*, all of which can look vastly different depending on the project or organization in focus when attempting to introduce RE or RM into their workflow.

5.1.1 Intrusiveness, Obtrusiveness and Clarity

In our experiment, T-Reqs can be characterized by its obvious obtrusiveness to the developers, but the intrusiveness within the workflow in DevOps varied between ICI and ICII since it went from only being a warning during each run to becoming a potentially blocking stage. However, these two factors can take many different shapes, especially between the three stages we have manipulated during this study, those being the process, RM and DevOps.

As mentioned in other studies in Chapter 2 and at the core of our problem space, these two factors are what make RE and by extension RM unattractive as a traceability option due to its diffuse interpretation, primarily when it comes to traditional RE concepts. With agile RE and RM, the approach taken is much less structured upon these preconceptions and you instead attempt to find the parts that can bring about the least overhead whilst also maximizing the benefits that RE offers.

This idea appears strong in theory, but due to the lack of research, clear definitions and widespread use, the migration to agile struggles to find a clear and substantial footing with modern development. When you move higher up along the value chain to the stakeholders that may differ in their technical knowledge, the confusing nature of RE brings even more problems as the understanding of *what needs to be done* and *what has been done* through an RE system would at many times require prior knowledge within the field. While this study mainly aims to look through a developer perspective, integrating RE and RM would often need to be unanimously agreed upon, and this first step may already cause many companies to look toward other solutions that involve a less steep learning curve.

These points can be seen from our research in ICI, where resistance was found within the difference between a need and a requirement and why it operates on the same level in the model. The granularity of these different types should be distinct enough in its technical aspect if they are within the same abstraction level, as the focus within an agile setting should be on distinct and straightforward functionality that makes for an easier understanding so that less time is spent on accurately designating if a request description should be classified as a need or a requirement. From the changes made and observations found for ICII, we can identify several ways to help alleviate these types of problems:

- **Simplicity before traceability:** In the case of this study, even if the model is based upon the most simple steps of a development process, its improvement on the traceability and feature tracking will be substantial enough to warrant its inclusion. Adhering to sets that are easy to recognize and separate from one another minimizes the organizational overhead and keeps the feature tracing as long as the core elements are still present. Favoring a vertical, hierarchical structure where you avoid horizontal elements that share the same level of granularity helps with granting clarity to developers. In this study, we explored the idea of consolidating types that may appear diffuse to developers. For example, we considered combining *stakeholder-need* and *stakeholder-requirement* into a single *stakeholder-requirement* type that depends solely on the granularity provided by the stakeholder. This approach aims to simplify the categorization and understanding of requirements based on stakeholder perspectives..

  Horizontal relations that share the same level of abstraction should only be present if their functionality is different enough to warrant its distinction. In agile development, the probable setting where this happens is within levels closer to developers such as in our case where we divided the low-level *feature* type to *code* and *config*.
- **Substitution before addition:** A problem that arose both due to the inexperience of RM and the existing bookkeeping documentation strategies was that the requirements many times looked like a carbon copy of the user stories stated for those features. Instead of blaming the developers for this result, we should look at interventions that can alleviate these issues. As mentioned

in a quote in Section 4.1.2.1, if the separation of granularity can be achieved between these two documentation efforts, you gain a much stronger sense of traceability within your project. However, this result still means that the RE stage of the development process is wholly an addition that takes more time away from creating these features and another point that may scare away potential adopters of agile RM.

One of our proposed solutions is to look at agile RM as a substitution for an agile storyboard rather than a completely new addition to the process. By creating requirements without having user stories in mind beforehand and later using the new additions to the model as the workload of a sprint (in the process then eliminating user stories/work cards), you can avoid the preset conceptions of a feature description from translating it to a user story. This keeps the obtrusiveness needed for the tool to be properly used yet eliminates much of the resistance that the initial introduction would otherwise produce. However, this type of solution is based on tool maturity and the ability to transfer the created low-level elements to storyboard-like features in existing DevOps services.

Another point to consider when discussing the obtrusiveness and intrusiveness of an RM tool (or any tool that includes a new process in development) is that these factors need to be noticeable to the subjects in order to make an actual impact yet gauged correctly so that the users of the tool use it correctly instead of finding ways to circumvent its time toll.

"*Developer: It is great that an academic study is integrated without breaking stuff, but some tools need to be 'in your face' to actually be used.*"

5.1.2 Tool Maturity and Flexibility

Even if the underlying idea and functionality of T-Reqs have proven to work in an industry setting in Ericsson (albeit not with the open-source release used in this study), the missing features and implementations added for ICII clearly show that it and by extension other agile RM tools lack in their range of application, as most are very inhibited by their ways to visualize its content and the difficulty that comes with developers being at the forefront of the RE and RM process. As mentioned in our first reflection meeting, there can be a certain danger with allowing the developer(s) to have all the power. The team in this study is already experienced with discussing the best strategy to fulfill a proposed stakeholder need and are very open to discussing any potential roadblocks or changes, but it would be naive to say that this is always the case. Other teams may have problems with miscommunication or misaligned goals in how you would want to create a feature or requirement when starting from the high-level abstraction. You could then look at solutions brought up in this study such as having product owners or team leaders that facilitate development and make sure that the team within the project have a unified understanding of what needs to be done and why.

However, this is where most RM tools can hinder such an approach as they often share the problem of having to be familiar with an IDE and git, along with terminal commands if you use T-Reqs. There are certainly people with this role who have this technical knowledge, but nevertheless, it becomes another hurdle that speaks against the implementation of these tools and RM as a whole. What then are the solutions to this new problem? The introduction of a simple yet effective GUI in this study has been received well, and these sorts of translations or ways to simplify the work done could be ineffective as a source of acceptance of agile RM. Another comparison for future ideas can be exemplified by the difference between the GitHub Desktop application and git, where simplicity and usability are in focus rather than the full suite of features for version control. We consider any solution that enables more people in the value chain to interact with and understand agile RM to be a good one, as one of the major faults of the field is seen in its obscurity within modern software development.

When looking at tools purely from a developer perspective, the need for extensive querying functionalities stands out as a very important one. With a model having several abstraction layers, in this case from the start of a stakeholder need onto the highest level of granularity in specific code of configurations means that a way to make effective queries that can be traced through these levels helps immensely. To do these in one line rather than having to navigate one level change at a time (much akin to navigating a traditional requirements document) means both better ease of use and the ability to quickly answer questions from stakeholder when it comes to showing how a project supports their needs. Optimally, this functionality would be available in the visualization efforts the tool provides, such as deriving the desired query from a visible tree that shows all elements found in the project.

Lastly, due to the hierarchical character of RE between its different elements, automation efforts should be targeted towards alleviating changes made to requirements already present within the project. The focus on detail does mean that changing one requirement or feature description due to e.g scaling changes or added features can result in the need to change parts across the whole project. If there exist features to both easily access all documentation connected to the changed element and apply changes to all affected elements at once, it would help with adapting agile RM further into the ever-changing agile environment.

5.1.3 Project and Team Characteristics

While it may come across as counter-intuitive to the aim of this study, there needs to be an understanding of when RE and RM actually is the best documentation alternative to a project. Due to the project structure involving a mono-repo, this aspect became very noticeable as time went on and requirements were created. The difference in how layered a project is in its technical aspect or even the difference in its parts should first be examined when looking at an agile RE solution.

Additionally, the predicted lifetime for the project and the importance of maintenance should also be looked at beforehand, as most RE approaches show their value the longer the project lives on.

In this study, the project involved both a technical platform with its accompanying infrastructure and an application which users interact with. Here, the differences of backend and frontend were found through the lens of RM, particularly in the difference of testing between the two. The characteristics of this type of platform works hand in hand with the strengths of RE and RM. Compliance with security requirements, a sub-set of non-functional requirements that have clear and visible metrics and the bias toward lower level testing are all factors that the RE approach benefits from. On the other end, a user application is harder to model and create requirements for, particularly so since many feature requirements and needs can be hard to pinpoint due to the subjective nature of a frontend where end users are the main source of them. Additionally, the non-functional requirements or quality attributes in these types of applications also become even harder to validate for the same reason.

No type of methodology or tools is the go-to for all software projects in the same vein that there is not one approach to agile and SCRUM that fits all teams. The first stage of deciding on whether to implement agile RE or not is then to look at the artefact you are trying to create and see if fits within the frames that agile RE operates in. It excels in validating security, stability and correctness demands, in describing the totality of a system through several abstraction levels yet lacks in parts such as measuring usability and handling artefacts of user-focused nature. This is in line with one of our three main factors reported in Section 4.3.4, that being the complexity of the project in focus. The project that was the subject of our interventions, does inhabit a high level of complexity due to its many different components and potential use cases. If RE and RM had been adopted earlier in the project as documentation practices, both researchers and the development team agree that it would have led to improved observability across the project. However, while beneficial within these areas, the complexity of the entire system would not be the decisive factor in approaching RE and RM as a solution. Rather, the critical factor lies in inter-team communication, which significantly affects their potential implementation.

While the development team is located in the same area and have been involved in the project since its inception, RE and RM efforts require the involvement of external stakeholders in the process and in that regard we notice that the high external distance between the team and the external parties as discussed in Section 4.3.4 impacted the adoption of RM efforts. The theory of distances in software engineering [7] identifies cognitive, semantic, and temporal distances among people, among other types of distances. According to Section 4.3.4 and in the case for the team, we argue that RM is beneficial for addressing challenges stemming from temporal, cognitive, and semantic differences within project teams. RM manages temporal differences by establishing clear timelines and maintaining version control, ensuring

that all team members have access to current information despite variations in activity schedules. To overcome cognitive differences, RM emphasizes collaboration and knowledge sharing through open communication and training, fostering a shared understanding among team members with diverse expertise levels. Additionally, RM promotes consistency in terminology and uses traceability to resolve semantic discrepancies between artifacts like requirements specifications and test cases, ensuring alignment and minimizing misunderstandings.

5.2 Threats to Validity

Limited generalizability may arise due to the unique organizational context, and resistance to change could impede the successful implementation of the intervention cycles. Resource constraints and potential disruptions to daily operations pose practical challenges, while measuring the short-term effects versus long-term impact may be complex. Despite these limitations, careful planning, stakeholder engagement, and phased implementation can enhance the validity and applicability of the experiment's findings. Additionally, the complexity of DevOps introduces unpredictability, potentially leading to unanticipated challenges during implementation.

5.2.1 Construct Validity

Performing an action-based field experiment on the topic of RM integration into a team's existing processes and DevOps practices entails inherent uncertainties regarding the construct itself. Firstly, trying out proposed obtrusive actions in a real organizational setting with a team possessing diverse technical expertise introduces complexity and variability that can impact the validity of measurement tools and interpretation of results for the entire team. This requires careful consideration of contextual factors influencing outcomes. Moreover, the iterative cycles of action research, where feedback from previous phases informs interventions in subsequent phases, are valuable for refining solutions but can also introduce bias or assumptions based on prior experiences or feedback, potentially limiting exploration of alternative approaches or identifying blind spots in problem framing. Lastly, balancing the aim of identifying generalizable factors applicable across teams, organizational structures, and projects with the context-specific nature of precise actions addressing specific technical or procedural aspects allowed by action research is essential for understanding the boundaries of applicability to other contexts. More on generalizability will be discussed later.
We utilized ChatGPT to correct grammar and syntax in certain sections of the paper, aiming to enhance consistency and clarity overall. However, it's important to note that the AI tool may not fully grasp the study's context or is familiar with the tech-stack we interact with, potentially leading to inaccuracies in grammar or phrasing. Despite our thorough review of all AI-suggested changes, there remains a chance of minor discrepancies affecting construct validity. While these potential issues are worth considering, they are unlikely to have a significant impact on the overall findings or conclusions of our study.

5.2.2 Threats to Internal Validity

While considering threats to internal validity for this field experiment, several aspects warrant attention. Firstly, the duration of the study, consisting of two intervention cycles spanning two sprints of three weeks, may not be optimal for fully adopting, developing, and acquiring relevant skills in requirements management by the study subjects. A longer timeframe would allow for more extensive execution and reflection on the individual intervention cycles, enabling a deeper and more realistic evaluation of the research questions. Additionally, since Swedish was used during reflection and evaluation meetings contrasted with the thesis being written in English, it could potentially introduce language-related nuances or translation issues. Furthermore, the Hawthorne effect, where participants may alter their behavior due to awareness of being observed, cannot be discounted as a factor influencing study outcomes [15]. This is evidenced by the team's continued use of the interventions introduced in ICI even after the intervention cycle had ended, potentially indicating a higher level of tool acceptance.

5.2.3 Reliability

When concerning the reproducibility of this study, reliability issues can be found within two instances. Firstly, the choice of using reflection meetings that involve open-ended questions along with added discussion points based on the answers stated can prove hard to replicate. Furthermore, these meetings were done in Swedish to then be later transcribed and translated to English, a process that potentially loses some of the linguistic nuances to the answers. However, we deem the choice of this data gathering very valuable for the study as it produces rich discussion points between team members that would not have been collected otherwise. Additionally, we have described our goals with each meeting in detail along with providing the planned questions for each one to help with reproducing similar results.

The other problematic area in terms of reliability is the potential information lost due to privacy and security concerns such as not being able to reveal the full transcript of meetings or withholding more specific information about the team members or the project characteristics. We would deem that the changes made due to this constraint has not been large enough to affect the final results in any way, but they could possibly be a hindrance when trying to understand the full context of the study when attempting to reproduce a similar experiment.

A factor that strengthens the reliability of the research is within the case participants, as all team members had little to no experience with RE or RM, a scenario that mirrors most software development teams in the industry. Furthermore, the existing tools used for DevOps and Cloud purposes are widespread and used in many other projects, further ensuring that the case description is easily reproduced across future studies.

While the tool introduced in the study (T-Reqs) is considered niche, the findings made from using it has been evaluated and chosen so that they are not specific to the usage of the tool to make sure that the study can be applied across other RE and RM implementations.

5.2.4 Ethical concerns

There have been no issue or particular attention given to ethical concerns for the duration of this study, you can however point to some areas that may need to considered from an ethical standpoint if this study were to later be used in an industrial or academic context.

In this paper, we have mentioned the role that automation has on developer acceptance and ease-of-use for RM tools. However, attention needs to be drawn to how you should ensure validation if you trust this to automated processes. In particular, if used within an industry with a large focus on compliance and safety/security, a fault in these tools and/or discrepancy in ensuring the correctness of the tools can prove to bring major repercussions. Additionally, it brings further ethical questions as to who is at fault when these problems are found. These ethical dimensions have not been in focus during this study, but should nonetheless be considered when attempting to implement any sort of automated tool into development processes.

5.2.5 Generalizability and Threats to External Validity

The generalizability of the findings from this field experiment in integrating RM tools, practices and processes in an agile team is crucial for understanding the broader implications and applicability of the study's results beyond the specific context in which the experiment was conducted. In this section, we discuss the factors influencing generalizability and the extent to which the findings can be applied to other teams and companies.

The outcomes of this experiment were influenced by several contextual factors inherent to Combitech AB and the team's structure and the nature of the project. These factors include:

- **Organizational Culture and Consultancy Practices** The experiment was conducted within Combitech AB, a medium-sized IT consultancy firm specializing in software development services. As a consultancy firm, Combitech's organizational culture emphasizes client engagement, rapid adaptation to diverse project requirements, and a collaborative approach with stakeholders. Consultancy firms like Combitech often prioritize flexibility and responsiveness to client needs, which can influence the feasibility and effectiveness of adopting new practices such as Requirement Engineering and Management. The consultancy environment may differ significantly from product-based companies in terms of project variability, client interactions, and project team dynamics. Therefore, the transferability of findings from this experiment to other differ-

ent organizational contexts should be considered with respect to the unique consultancy practices and project engagements characteristic of this industry sector.

- **Team Composition and Dynamics**:The experiment was conducted with an in-house project team at Combitech, which maintains close collaboration with client representatives. The team operates with flexibility for working from home while emphasizing in-person interactions and maintaining the same time-zone and language alignment with the client. The accumulated team experience spans over two decades, providing a wealth of expertise in software development and agile methodologies. The composition and dynamics of this project team significantly influenced the outcomes of the experiment. Team cohesion, facilitated by regular in-person visits from client representatives, played an important role in shaping communication styles and collaboration approaches. The team's extensive experience in software engineering and agile practices contributed to the adoption and effectiveness of the RE/RM practices introduced during the experiment. It's important to recognize that team dynamics, communication styles, and skill sets within this specific project team may differ from those in other organizational contexts. Variations in team characteristics across different teams could impact the generalizability of the experiment's findings to other consultancy firms or software development teams operating under different circumstances.
- **Project Characteristics**: The experiment was conducted within a software project at Combitech AB, characterized by a mono-repository architecture housing multiple microservices components deployed in the cloud. This project not only encompasses software components but also manages configurations of the cloud infrastructure supporting these services. The mono-repo-based architecture and cloud deployment pose unique challenges and opportunities for introducing RE/RM practices. The project's scope, timeline, and specific requirements played a significant role in shaping the feasibility and effectiveness of the interventions. The interconnected nature of microservices within a mono-repo and the need to manage both code and infrastructure configurations highlight the project's complexity.

  It's important to recognize that the project's architecture and technical characteristics may differ across different software development environments. Variations in project scope, technology stack, and deployment infrastructure can influence the applicability and generalizability of the experiment's findings to other software projects, particularly those adopting similar mono-repo and cloud-based architectures.

5.2.6 Generalizability Argument

To assess the generalizability of our findings, we consider the study-subject of the field experiment's *similarity to other organizations and projects*: According to Statista, Swedish enterprises are increasingly leveraging IT consulting and implementation services to enhance operational efficiency and maintain competitiveness. Emphasizing sustainability and effectiveness. Moreover, there is rising interest in

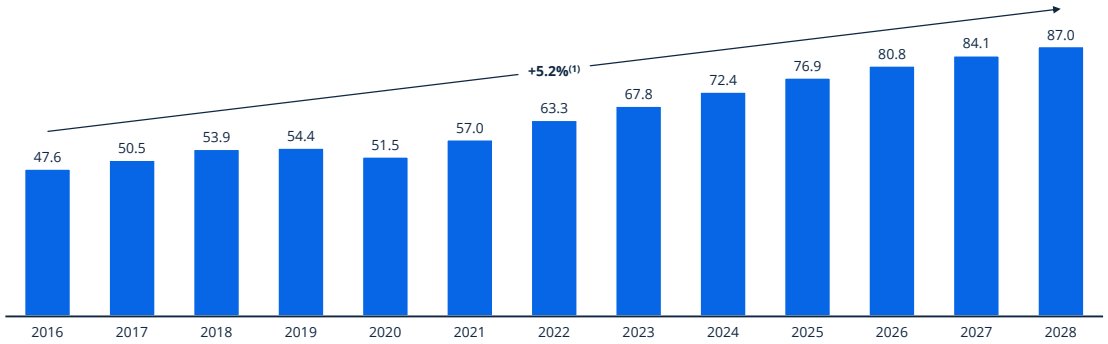


Figure 5.1: Estimated IT consulting revenues (in billion USD) from 2016 to 2028, showing a compound annual growth rate of 5.2% [2].

cloud-based solutions [1]. This it not unique to Sweden as Figure 5.1 shows a growing global trend in IT Consultancy revenues. Moreover, The field experiment's project aligns with the global trend of increasing adoption of cloud computing and the imperative for businesses to optimize their IT infrastructure, contributing to the growth of the IT consulting and implementation market [2].

We have demonstrated that the study subject of the field experiment, conducted in a real-world setting with a high degree of realism [32], belongs to the broader theoretical population within the context of software engineering. Consequently, we argue that our findings have merit for generalizability to a wider population encompassing a growing subset of similar software engineering projects. The insights gained from our study offer valuable qualitative evidence that can inform strategies applicable to similar projects within consultancy organizational contexts [34].

6
Conclusion

Within agile software development teams, the benefit of extensive documentation and detailed traceability within projects is often disregarded in favor of focusing on fast-paced and value-driven feature implementation during each sprint. As development teams and projects grow in size, the complexity and lifetime of these projects need this traceability to inform stakeholders of the capabilities within the system and why they have been created. To that end, agile Requirements Management has been looked at as a possible solution, a field that is at many times confusing to most modern software developers.

By examining various literature regarding this topic, there exists a need for more research on how to implement agile RM correctly within developer teams. This is due to challenges inherent to agile such as the many different implementations of it and the limited tooling support for agile RM. This study aims to find out the different factors that control the acceptance or resistance toward this documentation and traceability implementation, along with how to most effectively adapt it into a DevOps setting within an existing agile workflow. This has been done in a field experiment by implementing T-Reqs, an open-source requirements tool into a real-world setting that has then been evaluated based on two different intervention cycles. These cycles have produced a list of challenges, benefits, acceptance and resistance factors, and lastly observations about implementing RM into agile processes. The information gathered has been discussed with experts within the field of RE and RM to bring further value and generalizability to the findings.

Among the many different items and focus points found, the study finds that three factors are key when examining the value of RM within agile software development, those being the internal distance in teams, external distance from stakeholders and the complexity of the project. These factors have been validated by experts along with those involved in the field experiment.

While the study has provided many different insights into how agile RM and its benefits can be implemented within a software development process, the value of the implementation could not be fulfilled to its full extent since RE and RM often show its clear benefits when used across several years or when implemented at the start of a project. However, this setting also provides further insight into the problems that can arise if these documentation efforts are absent from a project.

During the study, the need for future work within different settings was discovered. The same implementation used in this study could be targeted to intervene in a project with either a short lifetime beforehand or an entirely new one. It would also be beneficial to examine requirements engineering at runtime through the use of T-Reqs to model resources within automation systems such as Kubernetes clusters.

Bibliography

[1] It consulting & implementation - sweden. Statista `https://www.statista.com/outlook/tmo/it-services/it-consulting-implementation/sweden` last accessed 27 Apr. 2024.

[2] It consulting & implementation - worldwide. Statista `https://www.statista.com/outlook/tmo/it-services/it-consulting-implementation/worldwide` last accessed 27 Apr. 2024.

[3] Requirements management software | jama connect | jama software, 2024.

[4] D. E. Avison, F. Lau, M. D. Myers, and P. A. Nielsen. Action research. *Communications of the ACM*, 42(1):94–97, 1999.

[5] K. Beckers and et al. A pattern-based method for identifying and analyzing laws. In *Requirements Engineering: Foundation for Software Quality: 18th International Working Conference, REFSQ 2012, Essen, Germany, March 19-22, 2012. Proceedings*, volume 18, pages 256–262. Springer, 2012.

[6] H. V. H. BIRGITTA FELDÍS BJARKADÓTTIR. Automating feedback for requirement changes in agile systems development, 2022.

[7] E. Bjarnason, K. Smolander, E. Engström, and P. Runeson. A theory of distances in software engineering. *Information and Software Technology*, 70:204–219, 2016.

[8] J. Browning and R. Adams. Doorstop: Text-based requirements management using version control. *Journal of Software Engineering and Applications*, 07(03):187–194, 2014.

[9] P. Checkland and S. Holwell. *Action Research*, pages 3–17. Springer US, Boston, MA, 2007.

[10] O. Gotel, J. Cleland-Huang, J. H. Hayes, A. Zisman, A. Egyed, P. Grünbacher, A. Dekhtyar, G. Antoniol, J. I. Maletic, and P. Mäder. Traceability fundamentals. In J. Cleland-Huang, O. Gotel, and A. Zisman, editors, *Software and Systems Traceability*, pages 3–22. Springer, 2012.

[11] L. Gren and P. Lenberg. Agility is responsiveness to change: An essential definition. In *Proceedings of the Evaluation and Assessment in Software Engineering*, EASE '20. ACM, Apr. 2020.

[12] V. T. Heikkilä, D. Damian, C. Lassenius, and M. Paasivaara. A mapping study on requirements engineering in agile software development. In *2015 41st Euromicro Conference on Software Engineering and Advanced Applications*, pages 199–207, 2015.

[13] H. F. Hofmann and F. Lehner. Requirements engineering as a success factor in software projects. *IEEE software*, 18(4):58, 2001.

[14] I. Inayat, S. S. Salim, S. Marczak, M. Daneva, and S. Shamshirband. A systematic literature review on agile requirements engineering practices and challenges. *Computers in Human Behavior*, 51:915–929, 2015. Computing for Human Learning, Behaviour and Collaboration in the Social and Mobile Networks Era.

[15] D. Isaacs and D. M. Berry. Developers want requirements, but their project manager doesn't; and a possibly transcendent hawthorne effect. In *Workshop on Empirical Requirements Engineering (EmpiRE 2011)*, pages 37–44, 2011.

[16] B. Kane. Estimating and tracking agile projects. *PM World Today*, 9(5):1–15, 2007.

[17] R. Kasauli, E. Knauss, J. Horkoff, G. Liebel, and F. G. de Oliveira Neto. Requirements engineering challenges and practices in large-scale agile system development. *Journal of Systems and Software*, 172:110851, 2021.

[18] R. Kasauli, G. Liebel, E. Knauss, S. Gopakumar, and B. Kanagwa. Requirements engineering challenges in large-scale agile system development. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 352–361. IEEE, 2017.

[19] E. Knauss, G. Liebel, J. Horkoff, R. Wohlrab, R. Kasauli, F. Lange, and P. Gildert. T-reqs: Tool support for managing requirements in large-scale agile system development, 2018.

[20] D. Leffingwell and D. Widrig. *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise*. Addison-Wesley Educational, Boston, MA, Dec. 2010.

[21] G. Liebel and E. Knauss. Aspects of modelling requirements in very-large agile systems engineering, 2022.

[22] A. P. Muhammad, E. Knauss, O. Batsaikhan, N. E. Haskouri, Y.-C. Lin, and A. Knauss. Defining requirements strategies in agile: A design science research study. In D. Taibi, M. Kuhrmann, T. Mikkonen, J. Klünder, and P. Abrahamsson, editors, *Product-Focused Software Process Improvement*, pages 73–89, Cham, 2022. Springer International Publishing.

[23] R. I. Nekvi and et al. Impediments to requirements-compliance. In *Requirements Engineering: Foundation for Software Quality: 18th International Working Conference, REFSQ 2012, Essen, Germany, March 19-22, 2012. Proceedings*, volume 18, pages 30–36. Springer, 2012.

[24] M. Pichler, H. Rumetshofer, and W. Wahler. Agile requirements engineering for a social insurance for occupational risks organization: A case study. In *14th IEEE International Requirements Engineering Conference (RE'06)*, pages 251–256, 2006.

[25] B. Ramesh, L. Cao, and R. Baskerville. Agile requirements engineering practices and challenges: an empirical study. *Information Systems Journal*, 20(5):449–480, 2010.

[26] B. Ramesh and M. Edwards. Issues in the development of a requirements traceability model. In *[1993] Proceedings of the IEEE International Symposium on Requirements Engineering*, pages 256–259, 1993.

[27] J. Savolainen, J. Kuusela, and A. Vilavaara. Transition to agile development - rediscovery of important requirements engineering practices. In *2010 18th IEEE International Requirements Engineering Conference*, pages 289–294, 2010.

[28] E.-M. Schön, D. Winter, M. J. Escalona, and J. Thomaschewski. Key challenges in agile requirements engineering. In *Agile Processes in Software Engineering and Extreme Programming: 18th International Conference, XP 2017, Cologne, Germany, May 22-26, 2017, Proceedings 18*, pages 37–51. Springer International Publishing, 2017.

[29] E.-M. Schön, D. Winter, M. J. Escalona, and J. Thomaschewski. Key challenges in agile requirements engineering. In H. Baumeister, H. Lichter, and M. Riebisch, editors, *Agile Processes in Software Engineering and Extreme Programming*, pages 37–51, Cham, 2017. Springer International Publishing.

[30] G. Spanoudakis and A. Zisman. Software traceability: A roadmap. *Handbook of Software Engineering and Knowledge Engineering*, 3:395–428, 08 2005.

[31] S. Stavru. A critical examination of recent industrial surveys on agile method usage. *Journal of Systems and Software*, 94:87–97, 2014.

[32] K.-J. Stol and B. Fitzgerald. The abc of software engineering research. *ACM Transactions on Software Engineering and Methodology*, 27(3):1–51, July 2018.

[33] N. Theodórsdóttir, Auður Katarína Ojensa. Evaluation of text–based requirements engineering tools, 2022.

[34] R. Wieringa and M. Daneva. Six strategies for generalizing software engineering theories. *Science of Computer Programming*, 101:136–152, 2015. Towards general theories of software engineering.

A

Type and Trace Information Model for First Intervention Cycle

TTIM in T-Reqs YAML format

```
---
name: T-Reqs Type and Trace Information Model (TTIM), 1st Version CT
version: 0.0.2
description: An initial model for the CT Meta Model
author: Abdullatif AlShriaf, Oscar Wallin
types:
  - name: stakeholder-need
    links: []
  - name: stakeholder-requirement
    links:
      - type: derivedFrom
        target: stakeholder-need
      - type: relatesTo
        target: stakeholder-requirement
  - name: requirement
    links:
      - type: addresses
        target:
          - stakeholder-need
          - stakeholder-requirement
      - type: hasParent
        target: requirement
  - name: component
    links: []
  - name: feature
    links:
      - type: relatesTo
        target: function
      - type: partOf
        target: component
        required: "true"
      - type: addresses
        target: requirement
        required: "true"
```

B

Requirement Checks Pipeline Stage

A `git diff` of the changes made to add the *Requirement Checks* stage

---

```
diff −−git a/azure−pipelines.yml b/azure−pipelines.yml
index a30694f3..116424c7 100644
−−− a/azure−pipelines.yml
+++ b/azure−pipelines.yml
@@ −11,18 +11,65 @@ resources:
 variables:
 [REDACTED]
+ # storageAccountBlob: '[REDACTED]'
+ storageAccountServiceConnection: [REDACTED]
 [REDACTED]
+ # TReqs + DevOps variables
+ treqsImageRepository: 'treqs'
+ treqsImageTag: 'IC1'

 [REDACTED]

 stages:
+− stage: RE
+ displayName: Requirement Check
+ jobs:
+ − job: TTIM_Check
+ displayName: Type and Traceability Model Check
+ pool:
+ vmImage: $(vmImageName)
+ steps:
+ − task: Docker@2
+ displayName: Docker login
+ inputs:
+ command: login
+ containerRegistry: $(dockerRegistryServiceConnection)
+ − task: Bash@3
+ displayName: Pulling TReqs
```

```
+ inputs:
+ targetType: inline
+ script: |
+ docker pull $(containerRegistry)/$(treqsImageRepository):$(treqsImageTag)
+ − task: Bash@3
+ displayName: Run TReqs Check
+ inputs:
+ targetType: inline
+ script: |
+ docker run −−read−only −v $(pwd):/usr/iiot −w /usr/iiot −t $(
    containerRegistry)/(treqsImageRepository):$(treqsImageTag) check −−
    verbose || echo "##vso[task.logissue type=warning]The model is inconsistent."
+ − task: Bash@3
+ displayName: Collect metrics
+ env:
+ TREQS_IMAGE: $(containerRegistry)/$(treqsImageRepository):$(
    treqsImageTag)
+ inputs:
+ filePath: $(Build.SourcesDirectory)/metrics/collect_metrics.sh
+ − task: AzureCLI@2
+ displayName: Storing metrics
+ inputs:
+ azureSubscription: $(storageAccountServiceConnection)
+ scriptType: 'bash'
+ scriptLocation: 'inlineScript'
+ inlineScript: |
+ az storage blob upload −f '$(Build.SourcesDirectory)/no_elements_in_model.
    csv' −c 'metrics' −n 'no−elements−$(Build.BuildId).csv' −−account−name '
    exjobb'
+ az storage blob upload −f '$(Build.SourcesDirectory)/no_links_in_model.csv'
    −c 'metrics' −n 'no−links−$(Build.BuildId).csv' −−account−name 'exjobb'
+ az storage blob upload −f '$(Build.SourcesDirectory)/no_inlinks_per_type.csv
    '−c 'metrics' −n 'no−inlinks−per−type−$(Build.BuildId).csv' −−account−
    name 'exjobb'
+ az storage blob upload −f '$(Build.SourcesDirectory)/no_outlinks_per_type.
    csv'−c 'metrics' −n 'no−outlinks−per−type−$(Build.BuildId).csv' −−account
    −name 'exjobb'
```

C

Metrics Collection Scripts

Shell scripts responsible for collecting metrics about the model

---

```
#!/bin/env bash

set -x
source $(pwd)/metrics/settings.sh

echo "element-type;count"
$TREQS_PREFIX list &> >(tail -n +3 | cut -d "|" -f 3 | sort | uniq -c | awk
    -v OFS="" '{print␣$2,";",$1}')
```

```
#!/bin/env bash

set -x
source $(pwd)/metrics/settings.sh

echo "inlink;count"
$TREQS_PREFIX list --inlinks &> >(grep "inlink" | cut -d "|" -f 3 | sort |
    uniq -c | awk -v OFS="" '{print␣$2,";",$1}')
```

```
#!/bin/env bash

set x
source $(pwd)/metrics/settings.sh

echo "element-type;no_outlinks"
mapfile -t types_array < <($TREQS_PREFIX list &> >(tail -n +3 | cut -d'
    |' -f3 | sort | uniq))

for type in "${types_array[@]}"; do
    type=$(trim $type)
    type_count=$($TREQS_PREFIX list --outlinks --type $type &> >(grep
        "outlink" | wc -l))
    echo $type,$type_count
done
```

```
#!/bin/env bash
```

```
set x
source $(pwd)/metrics/settings.sh

echo "element-type;no_inlinks"
mapfile -t types_array < <($TREQS_PREFIX list &> >(tail -n +3 | cut -d'
    |' -f3 | sort | uniq))

for type in "${types_array[@]}"; do
    type=$(trim $type)
    type_count=$($TREQS_PREFIX list --inlinks --type $type &> >(grep
        "inlink" | wc -l))
    echo $type,$type_count
done
```

D

Reflection Meeting Structure - ICI

Introductory & General Evaluation Questions

- Have you encountered any problems during ICI? - **RQ2, RQ1**
- How has your experience been during the cycle? - **RQ2, RQ1**
  - To what degree did you interact with the implementation during the cycle?
- Have you experienced an improvement in traceability? - **RQ1**
  - Do you see the potential for such an improvement based on the implementation made during this cycle?
- What would you say is the main benefit of this traceability, is it more beneficial to developers or other stakeholders, or both? **RQ1**

Requirement Meta-Model Questions

- How well did the meta-model capture your domain? - **RQ1, RQ2**
- Would you add links/types to the meta-model? - **RQ1, RQ2**
- Would you remove links/types from the meta-model? - **RQ1, RQ2**
- Were mandatory links constraining? - **RQ1, RQ2**

Process Questions

- Was the RE stage in DevOps obtrusive? - **RQ2**
- After updating the model, do you check model consistency locally? - **RQ3**
- Under which Agile moment(daily, backlog, retro) did you update the model (created elements and links)? - **RQ2, RQ3**
- How easy, would you say, is it to get an overview of what needs to be done/has been done? - **RQ2, RQ3**
- Did your current process and workflow help to ease the integration of RM made during the cycle? - **RQ2, RQ3**

Tool Maturity & Future Improvements Questions

- How was your interaction with the tool? - **RQ2**
  - How would you rate its usability and performance?
  - What other features would you like to see implemented into TReqs in order to facilitate the RM process?
  - In its current state, do you deem it a viable tool for RM and why?

  - Does TReqs help with your acceptance to RM by having the requirements closer to the developer?
  - Would the standardization of requirements help further with the RM process?
  - Is the current visualization in Markdown sufficient enough?
  - Would showing the traces and links within your project to other shareholders be of interest?

Closing Questions

- Are there any other questions or points you would like to discuss?

E

Type and Trace Information Model for Second Intervention Cycle

TTIM in TReqs YAML format

```
name: T−Reqs Type and Trace Information Model (TTIM), 1st Version CT
version: 0.0.3
description: An initial model for the CT Meta Model
author: Abdullatif AlShriaf, Oscar Wallin
types:
  − name: stakeholder−need
    links: []
  − name: stakeholder−requirement
    links:
      − type: derivedFrom
        target: stakeholder−need
  − name: technical−requirement
    links:
      − type: addresses
        target: stakeholder−requirement
      − type: hasParent
        target: technical−requirement
  − name: component
    links: []
  − name: code
    links:
      − type: relatesTo
        target: code
      − type: partOf
        target: component
        required: "true"
      − type: addresses
        target: technical−requirement
        required: "true"
  − name: config
    links:
      − type: relatesTo
        target: config
      − type: addresses
        target: technical−requirement
        required: "true"
      − type: partOf
```

target: component

F

Pipeline Report Markdown Template

Markdown template using Jijna engine

```
| Name | Type |
|--------------------|---------|
{%- for entry in entries %}
| [{{ entry.label_stripped }}](#{{entry.uid}}) | {{ entry.type }} |
{%- endfor %}

<hr/>

{% for entry in entries %}
<div id="{{ entry.uid }}">

### {{ entry.type }}: {{ entry.label_stripped }}
ID: {{ entry.uid }}

[View on Azure](https://dev.azure.com/cab-opiebe/_git/IIOT%20Platform?path
    ={{ entry.filename }}&version=GB{{GIT_BRANCH}}&line={{ entry.line
    }})

#### Outlinks

{%- for outlink in entry.outlinks %}
- {{ outlink.tlt }}  [{{ outlink.target.label_stripped }}](#{{ outlink.target.uid
    }}) of type _{{ outlink.target.treqs_type | lower }}_
{%- else -%}
: No outlinks!
{%- endfor %}

#### Inlinks

{%- for inlink in entry.inlinks %}
- {{ inlink.tlt }}  [{{ inlink.source.label_stripped }}](#{{ inlink.source.uid }})
    of type _{{ inlink.source.treqs_type | lower }}_
```

```
{%− else −%}
: No inlinks!
{%− endfor %}
</div>

<hr/>

{%− endfor %}
```

G

Pipeline Report Generation Task

Azure DevOps Bash task for generating and publishing the report

---

```
- task: Bash@3
  displayName: Generate Requirements Report
  inputs:
    targetType: inline
    script: |
      sed -i 's#{{GIT_BRANCH}}#$(branchName)#g' $(Build.SourcesDirectory)/templates/
          report.md # fill in branch name
      docker run --read-only -v $(pwd):/usr/iiot -w /usr/iiot -t $(containerRegistry)/$(
          treqsImageRepository):$(treqsImageTag) report --index templates/report.md --
          verbose || echo "##vso[task.logissue type=warning]Unable to genreate report."
      echo "##vso[task.uploadsummary]$(Build.SourcesDirectory)/report/report.md"
```

H

Requirement Change Tracker

Helper Bash script implementing a requirement `diff` feauture

```
#!/usr/bin/env bash

readonly TREQS_TAG="IC2"
verbose=false
for arg in "$@"; do
    if [[ "$arg" == "--verbose" ]]; then
        verbose=true
        break
    fi
done

function changed_files_folder() {
    # Create temporary directories
    temp_dir_old=$(mktemp -d)
    temp_dir_new=$(mktemp -d)

    # Get the commit hash provided as an argument; default to HEAD if no
        argument is provided
    commit=${2:-HEAD}

    # Create temporary files to store changed files
    changed_files_raw=$(mktemp)
    changed_files=$(mktemp)

    # Get the list of changed files between the specified commit and the current
        HEAD
    git log --oneline $commit..HEAD | cut -d' ' -f1 | while IFS= read -r line;
        do
        git diff --name-only $line >> "$changed_files_raw"
    done

    # Sort and remove duplicates from the list of changed files
    sort -u "$changed_files_raw" > "$changed_files"
     if [ "$verbose" = true ]; then
        echo -e "\033[1m List of changed files:\033[0m"
```

```
        cat "$changed_files"
    fi

    # Copy the changed files to the temporary directory
    while IFS= read -r line; do
        mkdir -p "$(dirname "$temp_dir_old/$line")"
        git show "$commit:$line" > "$temp_dir_old/$line" 2>/dev/null

        mkdir -p "$(dirname "$temp_dir_new/$line")"
        cp "$line" "$temp_dir_new/$line"
    done < "$changed_files"

    # Copy specific file
    git show "$commit:ttim.yaml" > "$temp_dir_old/ttim.yaml" 2>/dev/null
    cp ttim.yaml "$temp_dir_new/ttim.yaml"

    # Generate and compare lists of dependencies
    cd "$temp_dir_old" && treqs list |& tail -n +3 | cut -d'|' -f 2 | sort -u >
        list.out
    cd "$temp_dir_new" && treqs list |& tail -n +3 | cut -d'|' -f 2 | sort -u >
         list.out
    cd "$temp_dir_new" && treqs list |& sort -u > list.out.raw

    # Perform comparison and output any additional dependencies
    echo -e "\033[1m List of added elements:\033[0m"
    diff --color -u "$temp_dir_old/list.out" "$temp_dir_new/list.out" | grep
        '^\+' | tail -n +2 | cut -d' ' -f2 | while IFS= read -r line; do
        grep -r "$line" "$temp_dir_new/list.out.raw"
    done
}


if [[ "$1" == "diff" ]]; then
    # Call your function here
    changed_files_folder "$@"
    exit # Exiting here if we've called the function, assuming you don't want to
        execute further commands
else
        docker run --read-only -v .:/usr/project -w /usr/project -t
            combitechk8scr.azurecr.io/treqs:${TREQS_TAG} $@
fi
```

I

Reflection Meeting Structure - ICII

Introductory and General Evaluation Questions

- Have you encountered any problems during ICII? - **RQ2, RQ1**
- How has your experience been during the cycle? - **RQ2, RQ1**
  - To what degree did you interact with the implementation during the cycle?
- Have you experienced an improvement in traceability compared to ICI? - **RQ1**

Requirement Meta-Model Questions

- Did the new meta-model better capture the domain compared to the last? - **RQ1, RQ2**
- Was the migration to the new model easy? - **RQ1, RQ2**

Process Questions

- Was the added blocking RE stage in DevOps helpful/useful? - **RQ2**
- How easy, would you say, is it to get an overview of what needs to be done/has been done? - **RQ2, RQ3**

Tool Maturity and Future Improvements Questions

- How would you rate the new features based on usability and performance? (treqs diff, treqs report) - **RQ2**
- In its current state, do you deem T-Reqs a viable tool for RM and why?
- If not, what are the areas that need improvement to make it a viable tool?
- Is the added visualization sufficient enough, what could get even better?
- Have you or do you see potential with using the report feature to share progress with external stakeholders? - **RQ2, RQ3**

Closing Questions

- Would you consider using the interventions made during this study in the future? - **RQ1, RQ2, RQ3**
- Would you be willing to look at RE and RM as a practice in other current and future projects? - **RQ1, RQ2, RQ3**
- Are there any other questions or points you would like to discuss?