CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Automating Feedback for Requirement Changes in Agile Systems Development

Master's thesis in Software Engineering and Technology

BIRGITTA FELDÍS BJARKADÓTTIR
HEIÐRÚN VALDÍS HEIÐARSDÓTTIR

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2022

Master's thesis 2022

Automating Feedback for Requirement Changes in Agile Systems Development

BIRGITTA FELDÍS BJARKADÓTTIR
HEIÐRÚN VALDÍS HEIÐARSDÓTTIR

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2022

Automating Feedback for Requirement Changes in Agile Systems Development
BIRGITTA FELDÍS BJARKADÓTTIR & HEIÐRÚN VALDÍS HEIÐARSDÓTTIR

Supervisor: Eric Knauss, Department of Computer Science and Engineering
Examiner: Regina Hebig, Department of Computer Science and Engineering

Master's Thesis 2022
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2022

Automating Feedback for Requirement Changes in Agile Systems Development
BIRGITTA FELDÍS BJARKADÓTTIR & HEIÐRÚN VALDÍS HEIÐARSDÓTTIR
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Context: Managing requirements in large-scale agile systems becomes harder and harder as companies and products grow in an ever-changing environment where quick and effective responses to change are highly important. While previous on automated analysis of requirements documents exists, there are no obvious ways to automatically validate changes in requirements in agile environments where multiple changes are made in parallel to each other. Meanwhile, automated unit and integration tests are essential for agile practices to keep the main branch of the workflow clean.

Objective: This thesis investigates what can be done for requirements when it comes to automation, specifically in terms of requirements management and feedback. This is done through the investigation of the main problems of requirements verification and validation in agile systems development, when changes are often made in parallel to each other, that can be addressed using automated feedback. Then, an exploration of which potential solutions can be incorporated to improve the quality of the requirements, when changes are made in parallel to each other, are suggested and evaluated by requirements engineering experts.

Method: When conducting the study, three cycles of design science research were applied. Each cycle consisted of a regulative cycle containing the following steps: problem investigation, solution design, solution validation, solution implementation, and evaluation. The data collection of the thesis consisted of interviews, workshops, a survey, and a literature review.

Conclusion: Multiple solutions to the discovered problems were identified and rated valuable by requirements engineering experts. The solutions were categorized as interdependency feedback, feedback on change history, language feedback, or technical feedback based on the evaluation of requirements that they provide. These solutions are provided in the form of guidelines for practitioners and researchers to apply the findings to their own tools. The guidelines are constructed from the solutions rated as valuable by experts during the research. Suggestions of the points in time, in git-based workflows, where they should be applied, to which receiver to send the feedback, and a discussion of abstraction levels are further included in the guidelines.

Keywords: Requirements Engineering, Requirements Management, Automation, Software Engineering, Requirements Feedback, Agile, Design Science Research

Acknowledgements

We want to extend our gratitude to our supervisor, Eric Knauss, for trusting us with this research topic and his guidance throughout the course of the thesis. We would like to thank Filip Lange for being of help and support throughout the entire process. We are likewise thankful to Hans-Martin Hein and Grischa Liebel for their insights, our examiner Regina Hebig, and the participants of all interviews, survey, and workshops. Then, we thank the Software Center for giving us an opportunity to present and evaluate our work. Finally, a special thanks to all our Gothenburg friends, especially Johanna, Josephine and Tim, for their endless support, work sessions and fun times!

Birgitta Feldís Bjarkadóttir & Heiðrún Valdís Heiðarsdóttir,
Gothenburg, June 2022

"*Change, we don't like it, we fear it. But we can't stop it from coming or we get left behind.*"

-Meredith Grey

Contents

List of Figures **xv**

List of Tables **xvii**

1 Introduction **1**
1.1 Statement of the Problem . . . 2
1.2 Purpose of the Study . . . 3
1.3 Significance of the Study . . . 3
1.4 Research Questions . . . 4
1.5 Outline . . . 5

2 Background and Related Work **7**
2.1 Background . . . 7
2.2 Related Work . . . 8

3 Research Method **11**
3.1 Design Science Research . . . 11
3.2 Iterative Research Based on Regulative Cycle . . . 11
3.2.1 Problem Investigation . . . 12
3.2.2 Solution Design . . . 12
3.2.3 Design Validation . . . 13
3.2.4 Solution Implementation . . . 13
3.2.5 Evaluation . . . 13
3.3 The Artifact . . . 13
3.4 Data Collection . . . 14
3.4.1 Interviews . . . 15
3.4.2 Workshops . . . 17
3.4.3 Survey . . . 18

4 The Artifact **19**
4.1 Problems . . . 19
4.2 Solutions . . . 19
4.3 Value of Solutions . . . 21

5 Findings **23**
5.1 RQ1 (Problem) . . . 23
5.1.1 Key Issues . . . 23

5.1.2 Timings and Receivers . . . 24
5.1.3 The Problems . . . 24
5.1.3.1 Change History . . . 24
5.1.3.2 Interdependencies . . . 26
5.1.3.3 Language . . . 27
5.1.3.4 Technical Feedback . . . 28
5.2 RQ2 (Solution Candidates) . . . 29
5.2.1 Interdependencies . . . 30
5.2.2 Change History . . . 32
5.2.3 Language . . . 35
5.2.4 Technical Feedback . . . 37
5.2.5 Receivers and Timings . . . 38
5.2.5.1 Receivers . . . 39
5.2.5.2 Timings . . . 40
5.2.5.3 Connecting Timings and Receivers . . . 42
5.2.6 Abstraction Levels . . . 47
5.3 RQ3 (Evaluation) . . . 48
5.3.1 Feedback Types . . . 48

6 Discussion **51**
6.1 Interpretations of the Findings . . . 51
6.1.1 Receivers . . . 53
6.1.2 Timings . . . 53
6.1.3 Abstraction Levels . . . 54
6.2 Implications to Practice . . . 54
6.3 Implications to Research . . . 55
6.4 Limitations . . . 55
6.4.1 Construct Validity . . . 55
6.4.2 Internal Validity . . . 56
6.4.3 External Validity . . . 57
6.4.4 Reliability . . . 57
6.4.5 Ethics . . . 58

7 Conclusion **59**

Bibliography **61**

A Artifact **I**

A Interview Guide 1 **XXXV**
A.1 Introduction . . . XXXV
A.2 Introductory Questions . . . XXXV
A.3 Main Questions . . . XXXV
A.3.1 Change History of Requirements . . . XXXV
A.3.2 Types of Requirements Feedback . . . XXXVI
A.3.3 Quality of Requirements . . . XXXVI
A.3.4 Completeness of Requirements . . . XXXVI

A.3.5 Interdependencies between Requirements . . . XXXVI
A.3.6 Consistency of Requirements . . . XXXVII
A.3.7 Prioritization of Requirements . . . XXXVII
A.3.8 Prototype Questions . . . XXXVII
A.4 Extras . . . XXXVIII
A.5 Conclusion . . . XXXVIII

B Interview Guide 2 **XXXIX**
B.1 Introduction . . . XXXIX
B.2 Introductory Questions . . . XXXIX
B.3 Main Questions . . . XXXIX
B.3.1 Receivers of Feedback: . . . XXXIX
B.3.2 Change History: . . . XL
B.3.3 Interdependencies: . . . XL
B.3.4 Language: . . . XLI
B.3.5 Technical Feedback: . . . XLI
B.3.6 Feedback for... . . . XLI
B.3.7 Timings of Feedback: . . . XLII
B.3.8 Abstraction Levels: . . . XLII
B.3.9 Additional Questions: . . . XLII
B.4 Conclusion . . . XLII

C Survey **XLV**

List of Figures

3.1 The regulative cycle inspired by [31], altered to fit the thesis process. 12

4.1 Overview displaying the main problems of requirement verification and validation in agile systems that can be addressed using automated feedback, based on problem investigation through interviews and literature. 20
4.2 An example showing how user stories are formulated and displayed in the guidelines. 20
4.3 Matrices that show the business value in relation to the cost of implementing user stories, as rated by requirements experts. Larger figures displaying the matrices can be found in Appendix A. 21

5.1 Correlation matrix that shows the problems of requirements verification and validation on the horizontal axis, listed by their ID, and whether they can be solved by interdependency solutions that are on the vertical axis. 31
5.2 Correlation matrix that shows the problems of requirements verification and validation on the horizontal axis, listed by their ID, and whether they can be solved by change history solutions that are on the vertical axis. 33
5.3 Correlation matrix that shows the problems of requirements verification and validation on the horizontal axis, listed by their ID, and whether they can be solved by language solutions that are on the vertical axis. 35
5.4 Correlation matrix that shows the problems of requirements verification and validation on the horizontal axis, listed by their ID, and whether they can be solved by technical feedback solutions that are on the vertical axis. 37
5.5 Heat maps showing practitioners' ratings of specific receivers getting feedback at different timings for feedback types 1-6. Each row represents a timing (based on evaluation workshop in Cycle II) while each column represents a receiver of feedback. The values in the table represent the percentage of participants that approved of a receiver getting feedback at the corresponding timing. The darker the colour displayed in the table, the higher the correlation between a timing and a receiver for the relevant feedback type. 44
5.6 Heat maps showing practitioners' ratings of specific receivers getting feedback at different timings for feedback types 7-13. Each row represents a timing (based on evaluation workshop in Cycle II) while each column represents a receiver of feedback. The values in the table represent the percentage of participants that approved of a receiver getting feedback at the corresponding timing. The darker the color displayed in the table, the higher the correlation between a timing and a receiver for the relevant feedback type. 45

5.7 Heat maps showing practitioners' ratings of specific receivers getting feedback at different timings for feedback types 14-21. Each row represents a timing (based on evaluation workshop in Cycle II) while each column represents a receiver of feedback. The values in the table represent the percentage of participants that approved of a receiver getting feedback at the corresponding timing. The darker the colour displayed in the table, the higher the correlation between a timing and a receiver for the relevant feedback type. 46
5.8 Value-Cost matrix displaying all feedback types based on rating during a workshop with practitioners. The colors of the user stories indicate the feedback category they belong to (pink = Interdependencies, yellow = Change History, green = Language, and blue = Technical Feedback). 50

List of Tables

3.1 An overview of the data collection of each of the three cycles of the data science research, the steps of the regulative cycle where the method was used, number of participants and mapping to the three research questions. 15
3.2 A list of interviewees along with their role, their experience working with requirements and the cycle in which their interview was part of. 17

4.1 Business value and implementation cost of each of the proposed solutions, the feedback types, based on rating by practitioners during validation workshop. The ID represents the id of the feedback type the corresponding user story refers too. The category displays if the feedback type is categorized as Interdependencies (I), Change History (C), Language (L) or Technical Feedback (T). 22

5.1 Business value and implementation cost of each of the proposed solutions, the feedback types, based on rating by practitioners during final evaluation workshop. 49

1
Introduction

The expanding economy has caused several problems for large and growing organizations: most importantly, how to remain competitive and adaptable to the rapidly evolving market. The answer for many firms has been to use agile development methods [1]. The simplest way to define what agility means in the context of software systems is "responsiveness to change" [2]. Another common definition of agile software development methods is being more adaptive to changes than other traditional methods, where process and documentation are inferior to customer collaboration and working software [3]. Planning continuously and flexibly is a vital part of agile processes and task sharing is also a recurring process within agile [4]. These agile ways of planning are achieved more easily in small to medium-sized projects since in those cases planning is more likely to be short-term [3]. The question remains of how agility can be achieved in large-scale software projects.

Lack of research on large-scale applications of agile within requirements engineering has been a hindrance when applying agile to larger projects, especially when it involves larger systems that consist of more parts than only software, since such projects often have high demands when it comes to security and reliability [5].

A case study by Kasauli et al. [5] found that maintaining agility in scaled projects is often difficult, especially when the product relies on multiple cross-functional teams. Managing and updating requirements should follow a flexible process in order to maintain agility and keeping documents up to date is vital when avoiding confusion within an organization's teams. To counteract this problem, a suggestion of the case study was to incorporate a system into an organization's workflow, allowing for management of system requirements in the same version control system as source code and tests, thus allowing for cross-functional teams to more easily collaborate on and manage requirements.

One such system is T-Reqs, a textual requirements management tool that aims to allow teams to manage requirements simultaneously with code and tests. T-Reqs supports agile-cross functional teams in being aware of system-level requirements and proposing updates for them in an efficient way [6]. However, challenges remain both in an industrial and open-source context, and its value has, as of now, not been proven in these contexts [6].

The goal of this thesis was to mitigate the aforementioned challenges by identifying which problems arise during requirements management in agile industrial environments that can be solved by automating requirements feedback. A further goal was to discover which of the solutions are feasible to solve those respective issues by evaluating them with experts.

The study investigated the aforementioned challenges and their solutions using

three-cycle design science research where problem investigation, design validation, and evaluation were conducted through interviews, workshops, and a survey with practitioners and researchers. Solution design and implementation were based on the evaluation of data from the aforementioned data collection methods and then collected into an artifact on the form of guidelines for both practitioners and researchers. Those guidelines provide suggestions on how to apply the thesis findings to the respective context by presenting feedback types, timings of feedback, receivers of feedback, and discussions of abstraction levels.

The study is intended to benefit practitioners interested in expanding their tools and processes by providing them with guidelines, rated by experts, of automation of requirements feedback that can be applied in their environments. Further, the study benefits researchers by identifying the need for further investigation in the field of automating requirements feedback.

1.1 Statement of the Problem

In agile projects, requirements are volatile [5], [7]. The changes made to requirements are often made simultaneously by different teams using different processes and tools [5], requiring significant coordination efforts as different teams depend on each other's work [7]. This results in problems being introduced when writing requirements such as differences in the quality, completeness and prioritization of requirements [5] which introduces difficulties when working with the requirements. Those problems can introduce issues and inconsistencies of the final product, the consequences of which can be project failure and business loss [8]. Thus, continuously integrating multiple parallel changes to requirements undeniably requires a feedback mechanism.

The environment of multiple teams or developers in agile environments will inevitably introduce interdependencies among them [9], [6] which can incorporate inconsistencies in the changing of the requirements themselves, such as who should make the changes and at which points within the agile process they should be made [5]. Prioritization of requirements further introduces consistency issues between teams, as their perspective of criticality can be diverse [9].

New approaches to requirements engineering in scaled agile partially mitigate these problems. For example, tools that integrate textual requirements into version control systems such as git provide developers with access and peer-reviews of requirements through their regular infrastructure for continuous integration [6].

Additionally, lacking communication between teams causes synchronization issues among them, since having a large variety of stakeholders, teams and projects, as well as a large variety of organizational levels, introduces complexity [5]. The complexity issue that is caused by the lacking communication could be solved with increased access to the change history of other teams, simultaneously as changes are made.

Automating software testing is an important practice in software development, especially in agile environments. Automated tests provide multiple benefits such as rapidly accommodating changes and give the maximum amount of test coverage with minimal cost in regards to both time and money [10].

Continuous integration of code is important for agile practices in order to keep the main branch of the workflow clean and ready to deploy [11], but the question of what can be done for requirements remains.

The problem this thesis addresses is the fact that organizations struggle with managing changes made to requirements in agile environments where multiple changes are made in parallel to each other. To counteract that problem, this thesis investigates what solutions exist for automatically giving feedback on requirements and how they can be applied.

1.2 Purpose of the Study

The purpose of this study is to discover which types of automated requirements feedback prove to be the most beneficial when integrating multiple parallel changes to requirements in agile environments. This was done by investigating problems reported by practitioners and researchers, using the problems as ground for creating solutions on the form of feedback types that aim to solve the problems, and then evaluating the usefulness of the solutions in regards to solving the problems.

Moreover, the purpose is to investigate at which point in the workflow, i.e. the controlled process consisting of coordinated activities that execute an organization's business or process tasks [12], different feedback at different abstraction levels proves to be the most useful in relation to the quality of the requirements.

The textual requirements management tool T-Reqs was chosen as the baseline tool for the research for the reason that it supports solutions for requirements in agile environments using git. It allows for changing the requirements together with code and testing in iterations, making them more manageable for the development teams [13]. For those reasons it is straightforward to use the tool to integrate the new automatic feedback. Additionally T-Reqs is currently being used in industry, at a telecommunications company within the Gothenburg area, which allows for further involvement and integration in an industrial and large-scale agile setting.

Furthermore, the purpose of the research is to investigate whether there are common problems within requirements engineering that can be avoided by giving appropriate feedback through automation.

1.3 Significance of the Study

The study aimed to further knowledge of automated feedback in the git-based workflow of requirements in agile projects. Automated requirements feedback can provide higher quality of requirements in a shorter amount of time and allow for agile systems to maintain a clean branch, e.g. in regards to consistency and correctness, at all times regardless of multiple simultaneous changes. Additionally, the automated feedback allows for more concrete requirements which in turn increases the likelihood of correct interpretations of requirements before development. Requirements engineers can thus spend less time focusing on how to structure the requirements and spend more time focusing on the content of each requirement.

Knowing which feedback to give at which points in the git-based workflow has

the potential to make the automated feedback more concise and of greater quality, which increases the maintainability and quality of requirements in agile projects which additionally makes for easier interpretation of the requirements.

1.4 Research Questions

The integration of automated feedback into agile workflows can be valuable. This is for example visible in the claim of Post and Fuhr [14] that traditional requirements reviewing methods are often manual and take too long. Thus, suitable automated support is needed. This study answers the following research questions:

RQ1: *What are the main problems of requirements verification and validation in agile systems that can be addressed using automated feedback?*
The motivation for this question is to identify the main issues in regards to validation of requirements that can be solved by automating feedback.

RQ2: *What are potential solutions that can be used to incorporate continuously automated feedback to improve the quality of requirements in agile systems that have continuous parallel changes?*
The motivation for this question is to identify possible new or improved solutions to the issues identified in **RQ1**.

RQ3: *How valuable are the proposed solutions as rated by practitioners in regards to supporting common problems caused by a lack of automated requirements feedback in agile systems?*
The motivation for this question is to reflect on the issues (**RQ1**) and solutions (**RQ2**) found and reflect on the extent to which they solve the issues reported in the thesis.

1.5 Outline

The remainder of this thesis is arranged in the following manner:

Background and Related Work equips the reader with the underlying background and theory in order for them to understand the context of the research and its findings. It includes the findings of a literature review of the discussed topic from related work, which highlight some of the most prominent challenges of current requirements engineering processes within agile contexts. Moreover, an introduction of the baseline tool for this research, T-Reqs, is included.

Research Method explains the methodology of the chosen research method and how it was utilized when conducting this research to answer the three primary research questions.

The Artifact demonstrates how the challenges targeted by the research questions can be solved by demonstrating the solutions in guidelines that suggest ways to improve automatic validation of requirements, linking the solutions to the respective problems they aim to solve.

Findings reports the outcomes and findings of the research's cycles, i.e. the analysis of problems, solution candidate validation, and implementation and evaluation of the solutions.

Discussion contains a discussion of the outcomes of the research and its implications for research and practice.

Conclusion is a reflection upon the concluded research and the outlook for further research on the topic.

2
Background and Related Work

This chapter introduces the topics that are relevant to answer the research questions. Previous research and related tools are also established in order to explain the existing work related to the subject of the study.

2.1 Background

While agile requirements engineering processes have become somewhat the status quo for small to medium sized software projects [1], [3], applying agile processes to large-scale contexts has proven challenging [5], [6], [15]. As organizations strive to become and remain competitive, a common way of facilitating that process has been to use agile development methods, with the aim of minimizing the risks associated with software development [1]. However, there is a lack of research within requirements engineering when it comes to applying agile methods to real-world large-scale applications [5], [4].

Frameworks such as Scaled Agile Framework (SAFe®) and Large-Scale Scrum (LeSS) are being adopted to provide scaled agility in larger projects. SAFe® attempts to incorporate all best practices which in turn makes it very descriptive and thus is considered by many practitioners to be too bureaucratic to be agile, while LeSS is less definitive and allows for more freedom when it comes to adapting the framework to each context's needs [15].

Continuous integration is one link of the continuous practices that are common within agile processes, along with continuous delivery and continuous deployment. These practices are intended to provide faster feedback from processes and customers, more frequent releases of software and eliminating manual tasks [16]. More specifically, continuous integration is an established agile software development practice [10], [11], [16] in which code is merged frequently with the aim of shortening the release cycle and improving software quality [16]. Part of eliminating manual work is incorporating automatic testing into the workflow. Before new code features can be deployed in the continuous pipeline, they are tested for correctness as part of the continuous integration part of the pipeline [17].

Large systems rely on the underlying research in requirements engineering to support their large-scale agile development [6]. A crucial part of agile processes is the ability to plan continuously while being flexible to changes [4], i.e. responsiveness to change [2]. Planning continuously and flexibly is a vital part of agile processes as customer wants and needs change over time leading to changes in requirements [4], which can make continuous management of requirements a challenge [7].

This prompts the need for continuous validation of the product and its requirements [4]. Requirements validation is one of the biggest challenges of traditional requirements engineering since validation and verification are traditionally planned based on requirements [4]. When the development is agile the requirements are often incomplete and changing over time, thus following agile practices can mean less traceability between tests and requirements in addition to imprecise relations between information items [4].

Traceability is defined by Torkar et al. [18] as being the ability to follow and describe the life of requirements. They further recognize that traceability can both have a forward and a backward direction, that is, requirements can be related to their origin, i.e. stakeholders, business rules or previous documents, as well as other requirements before they are defined in a requirement specification, and requirements traceability can also be performed after the requirement is defined in a specification and in that case relates to a requirement being fulfilled such as by being traced to a test case.

In order to maintain agility among cross-functional teams in scaled projects, the management of requirements needs to be a flexible process [5]. When it comes to managing requirements, task sharing is a relevant topic as information regarding changes needs to be shared with minimal communication delay [4], [5].

2.2 Related Work

Traditional ways of manually reviewing requirements can take a long time based on agile standards [14], and although it is most common to check for errors manually it is very costly to do so as it is in fact one of the costliest phases of requirements engineering [19]. Getting real-time feedback on changes would therefore be a great improvement to this task [14].

One such automatic tool is IBM's Requirements Quality Assistant on which research by Post and Fuhr showed that it does not satisfy the needs of the selected domain, where measuring semantics such as consistency and completeness when writing requirements was considered more valuable than mostly just having checks for syntactical rules [14].

However, syntactical rules are in fact important when trying to achieve good quality in requirements [20]. Templates are therefore useful when it comes to synchronizing changes to requirements made by multiple agile teams [6], [20].

Most requirements documents, i.e. requirements specifications, are written in natural language [21], [22]. Natural language processing is a collection of computational techniques that analyze naturally occurring texts at one or more levels of linguistic analysis with the aim of achieving language processing for a range of tasks or applications, the same way as a human would process language [23].

As Zhao et al. [21] discuss in their paper, natural language is the most common notation in industrial settings for describing and specifying software and system requirements. They assume that natural language will in the future continue to be the dominating technique for expressing requirements.

The goal of natural language processing for requirements engineering is to support requirements engineers when they perform tasks such as detecting language problems

and identifying key domain concepts and establishing traceability links between requirements [21].

Similarly, a study by Ormandjieva et al. [19] came to the conclusion that when it comes to evaluating textual requirements it is possible to create an automatic decision-making emulation system that mimics the decisions a human reviewer would make, as long as the human annotators have agreed on a standard for the automatic classification of the quality of requirements.

In a case study by Kasauli et al. [5] a suggestion was made to incorporate a requirement validation system into the organization's workflow that would help with managing system requirements through a version control system, thus allowing requirements to be treated the same as code and tests for easier collaboration across teams. Code refactoring is an ongoing activity in agile projects and is a way of changing software without changing its existing behaviour in order to improve its architectural structure and accommodate changes [4], [24], so investigating how the same mechanics can be applied to changes in requirements could be beneficial when designing a requirement validation system.

Importantly, fully automating processes might not be a magic solution to the problems that exist within requirements verification and validation. A study by Dalpiaz et al. [25] found that despite having tool support, it is still time-consuming to identify terminological ambiguities in requirements. Determining whether a word has the same meaning as as another word is a challenge when it comes to making sure that a software system is being developed correctly, i.e. that the requirements are correct, so having an entirely automated system can therefore introduce new issues. Therefore, they suggest that it might be most effective to combine manual inspection with automated tools.

Machine learning efforts have been made when it comes to validating textual requirement reviews, such as in a study by Singh [26] where an approach is suggested to identify true and false positives when natural language processing is used to locate faults in requirement specification documents. Additionally, the study suggests approaches to identify interrelated requirements within a requirements specification and use natural language processing to help requirement authors write high-quality requirements.

Requirements traceability is generally seen as a costly branch of requirements engineering, and tools exist that intend to make tracing easier. Examples of common requirements management tools in industry that handle traceability between requirements are IBM's Rational RequisitePro and IBM's DOORS. Rational RequisitePro identifies affected software artifacts when changes are made to requirements, as it offers the ability to link to use-case diagrams and test cases. DOORS highlights changes that can potentially affect other requirements, and offers ways to create, trace and analyse links between requirements [18].

Other tools commonly used in industry are Jira [5] and SystemWeaver [27]. Jira is primarily an issue-tracking tool for software systems, and it supports error documentation and reporting as well as status report writing. It can also be used to customize workflows, that is, it is possible to treat a requirement as it if it was an issue and use the issue tracking functionality to assign a requirement to a person responsible for it. The responsible of a requirement can then manually set the state

of the requirement and connect it to a quality check, and that responsible is later notified if the requirement fails a quality check [28]. SystemWeaver is explained by Wohlrab et al. [27] a customizable requirements modeling tool that supports traceability between artifacts as well as providing visualisation of data. Among other features, it also shows when a requirement was created or changed and by whom. However, traceability is manual in SystemWeaver, thus it lacks support for detecting how changes to requirements affect tracelinks and updating them appropriately, for example in a semi-automatic way.

While these tools include a lot of desirable features for requirements management, they do not seem to evaluate the quality of requirements when it comes to their content. Additionally, it is unclear how well suited to agile standards these tools are [5].

One system intended to break down the barriers of applying agile methods to larger systems is the textual requirements management tool T-Reqs [5], [6], [13].

T-Reqs is a text-based requirements management tool that is managed using the version control system git. It addresses the challenges of large-scale agile development by incorporating agile conventions in templates and helper scripts in order to help cross-functional teams manage requirements and suggest changes to them [6].

The tool T-Reqs is currently under evaluation as an open-source system and is already being used in industry. Its solutions have been proven useful for requirements management in large-scale agile companies with a git-based infrastructure [13].

3
Research Method

The methodology used when conducting the study was Design Science Research. The study followed the guidelines of the Design Science Research Process Framework as described by Hevner [29], which provides guidelines for the understanding, execution and evaluation of the research. Further, the guidelines presented by Knauss [30] were used as reference for the process of conducting Design Science Research in the context of a master thesis.

3.1 Design Science Research

Design Science Research is a problem-solving method that aims to iteratively design useful artifacts by defining innovative ideas, practices, technology and products [29], which was seen as a beneficial method to create an artifact while keeping the process agile and open to changes that appear throughout the research process.

The selection of the chosen method was based on the goal of the thesis, which aims to analyze and suggest solutions to the problems that arise when simultaneous changes are made to requirements in agile systems. This goal aligns with the main principle of the design science research methodology to gain understanding and knowledge about a design problem and apply an artifact to the problem to get to a solution [29].

The aim was to achieve this goal by analyzing the problem with a case study, both in an industrial and non-industrial setting, and suggesting solutions in the form of a design artifact which could be applied to an agile system.

3.2 Iterative Research Based on Regulative Cycle

Iterations were conducted based on the regulative cycle, which is a cyclic problem solving process that goes through the steps of analyzing a problem, proposing solutions, and evaluating those solutions before selecting the most compatible and repeating the cycle, with the new solution as a starting point [31]. The regulative cycle was altered to fit the study as represented in Figure 3.1. The regulative cycle consists of four steps which were conducted based on the following descriptions.

Three repetitions of the regulative cycle were conducted during the research process, where each one included all steps of the cycle. However, iterations had different emphasis based on their position in time of the research. The first iteration focused on problem investigation, the second one on the solution design and finally the third one focused on evaluation of the proposed solution.

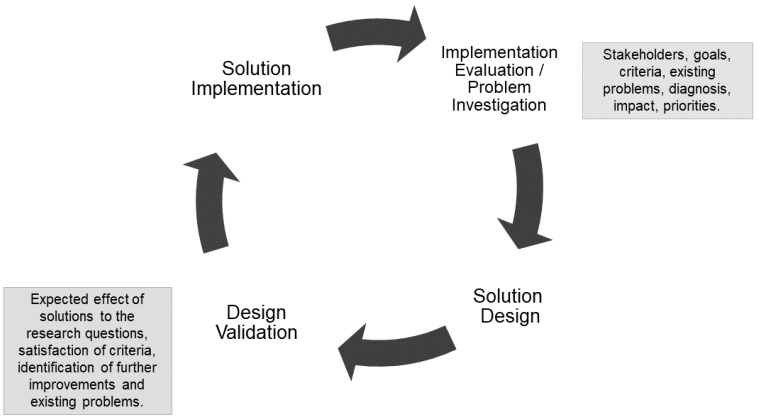


Figure 3.1: The regulative cycle inspired by [31], altered to fit the thesis process.

3.2.1 Problem Investigation

The investigation of the problem benefited from a large-system telecommunications company in the Gothenburg area that is currently using T-Reqs and similar tools in their system development, as well as other similar companies participating in a joint Software Center Project[1]. An initial workshop within Software Center was conducted in October 2021, prior to the formal start of the research process, which indicated interest in both the study topic and its relevance.

Problem investigation was carried out using various methods. During Cycle I, the problem investigation focused on literature along with a higher-level interview in which issues within the requirements validation process were discussed. Interviews with experts from both industry and academia were conducted as part of problem investigation of the following cycles. These individuals are core team members and key users of the open-source variant of T-Reqs as well as the industrial version of T-Reqs.

3.2.2 Solution Design

The solution was created based on findings of reported problems and solution suggestions from interviews conducted in each cycle. A difference in Cycle I was that the solution design was based on results from the previously mentioned initial workshop, along with a literature review of previous work, which included investigating similar tools and processes. For Cycle I the solution design was presented as a list of interview questions with the goal of eliciting ideas and validating findings from literature.

[1]https://www.software-center.se/

3.2.3 Design Validation

The design is validated with case study interviews in agile environments and workshops with experts who are part of the Software Center. The goal of the design validation was to identify successful solution candidates along with developing further improvements and problems.

The process of design validation was conducted by demonstrating the artifact during semi-structured interviews. During the interviews, investigators asked questions regarding the usefulness of each finding as well as encouraging interviewees to come up with further solutions of the previously discussed issues.

3.2.4 Solution Implementation

Solution candidates that were found promising during design validation were then applied to the solution of the research, which is the artifact of the design science research study. The solution implementation's purpose is to enhance the artifact based on findings from the previous steps of the cycles [29] with the aim of building the best version of the automated feedback concept.

3.2.5 Evaluation

The different evaluation phases utilized different evaluation methods. During the first two cycles, the evaluation coincided with the problem investigation phase of the next cycle. For Cycle I, semi-structured interviews were conducted. For Cycle II, a workshop with Software Center was used to evaluate the current findings at that point. For the last cycle, Cycle III, a workshop was arranged that was dedicated to evaluating the findings of the research by presenting and discussing the design choices and the resulting improvement suggestions to feedback systems for requirements in agile projects. This workshop was conducted with practitioners and researchers, both ones who had previously taken part in other steps of the research process and ones who had not taken part in previous steps.

3.3 The Artifact

The artifact of this design science research is on the form of guidelines for a requirements feedback system in agile projects. It focuses on which feedback types should be given, at which time and to whom.

The guidelines connect the proposed solutions, i.e. the automated feedback types, to the common problems caused by a lack of automated requirements feedback in agile systems. The guidelines also include timings in the git-based workflow where the feedback is most relevant, information about the receivers of each feedback type at each timing, and the abstraction levels of the feedback.

The purpose of the artifact is to help solve the discovered problems by giving suggestions on how to mitigate them. The proposed solutions are clearly connected to the corresponding problems that they aim to solve.

The artifact constantly evolved during the iterative regulative cycle, with the level of detail growing incrementally with each iteration of the regulative cycle until it had formed a constructive concept of a feedback system.

The focus of this research was to identify and find solutions to the main challenges of automating requirements verification and validation. Future possibilities for the artifact are to implement and integrate the solutions into either the open-source T-Reqs for demonstration of the solutions, or with the industrial version of T-Reqs which is already in use at a large telecommunications company in Gothenburg.

The artifact, on the form of guidelines, connects the findings of the thesis based on the research questions by combining potential solutions (**RQ2**) with the identified problems of requirements verification and validation in agile systems (**RQ1**). The artifact allows for putting the solutions into a tangible form that supports practitioners and academics using them in their own fields of work. Furthermore, it allows for the possibility of a concrete evaluation of the proposed solutions (**RQ3**) by clearly demonstrating them to practitioners and having them evaluate their usefulness.

3.4 Data Collection

At the beginning of Cycle I, problem analysis was conducted through a literature review and a higher-level interview with two T-Reqs experts. Later, data collection consisted of individual interviews with people working with T-Reqs in the software industry as well as discussions during workshops with researchers and industry experts. Finally, a survey was conducted as part of data validation of Cycle III. Table 3.1 provides an overview of the data collection of each of the three cycles of the data science research. The table further displays the steps of the regulative cycle that the method was used for and the number of participants. Finally, the table links the collected data to the research questions.

In total, the data collection consisted of a literature review, seven interviews, three workshops and replies from twenty survey participants.

An overlap is in the participants of the different activities. The two participants of the focus group interview were also interviewed as part of Cycle I's problem investigation phase, and one of them also participated in the evaluation workshop of Cycle II. All of the interviewees from the evaluation interviews of Cycle I also participated in the evaluation workshop of Cycle II. Further, the survey was sent out to all previous participants of the study in addition to individuals who had not previously been involved. Since the survey was anonymous and answers are not traceable to participants it is unknown how many of the respondents had already taken part in the study. This concludes that the minimum amount of participants of the study is 20 individuals, if it is assumed that all previous participants took part in the survey. However, there is high likelihood the number of participants is higher.

Participants for interviews, workshops and the survey were recruited through convenience sampling in order to collect information from reachable individuals that are knowledgeable and interested in the topic of the study [32]. The participants were typical users of T-Reqs and thus able to provide valuable feedback and information to the research. The participants recruited were from the previously mentioned

Table 3.1: An overview of the data collection of each of the three cycles of the data science research, the steps of the regulative cycle where the method was used, number of participants and mapping to the three research questions.

Cycle	Method	Step	Participants	RQ
I	Literature Review	Problem Investigation	-	1
I	Interviews/Focus Group	Problem Investigation/Design Validation	2	2
I/II	Interviews	Evaluation/Problem Investigation	3	3
II	Interviews	Design Validation	3	1 & 2
II/III	Workshop	Evalution/Problem Investigation	8	3 & 1
III	Survey	Design Validation	20	2 & 3
III	Workshop	Evaluation	8	3

telecommunications company that uses T-Reqs for requirements management, as well as developers and individuals with extensive understanding about the open-source variant T-Reqs. These individuals are all knowledgeable of requirements management and maintenance. Roles of participants of interviews were collected, but are not disclosed in detail in order to maintain their anonymity. Further, participants of workshops and the survey, as well as the interviews, were all recruited as individuals that are requirements experts and knowledgeable of requirements management.

Additionally, snowball sampling was used when recruiting participants for the independent workshop held by the researchers, as well as for the conducted survey, with the purpose of getting higher participation of experts interested in working with requirements. This was done by encouraging the invited participants to identify and reach out to their expert companions, who might have similar knowledge and interest about the study topic, for participation [32].

3.4.1 Interviews

Interviews were conducted as the main input of data in regards to problem investigation, design validation and evaluation. Interviews are the most common tool for conducting qualitative research, where the focus is on the interviewee's experiences, views and beliefs in regards to the research topic [33].

The first interview, which was conducted as a focus group, was part of Cycle I with a focus on problem investigation to better define the thesis's scope. This interview was unstructured, meaning it did not have a predefined questioning framework and the interview's flow was according to the direction the discussions between interviewers and interviewees took [33]. An additional goal was to identify which direction of research would be the most valuable both for industry and research. Further, the purpose was to get insights regarding which questions might be useful to ask in later interviews and get suggestions of individuals that would be useful to reach out to for further interviews.

The topics discussed during the interview were the thesis proposal and the scope of the project, as well as questions in the interview guide and other topics that emerged during discussions. These topics were discussed as part of investigating the problems of requirements verification and validation, as well as validating that the topics and questions asked in the interview guide were relevant for finding solutions to the problems.

The two external participants were selected based on their expertise and involvement in the T-Reqs project, one coming from an industrial background and the other from an academic background. Most research on the T-Reqs project is academic, while those with experience in its usage are from industry. It was thus considered a good starting point to initially get input from a mixture of these two branches of the tool, while in the future the emerging results would be validated by people who work with the T-Reqs tool or other requirements management systems.

The second round of interviews was conducted as part of Cycle I evaluation in overlap with the problem investigation of Cycle II, so they both served the purpose of collecting data for design evaluation as well as gathering new ideas for the next iteration.

These interviews, and all remaining interviews, were semi-structured and, as such, followed a structured interview guide but were flexible in regards to the interviewee and interviewers being allowed to stray from the interview guide. This method presented the opportunity for open discussions of any emerging topics raised by the interviewee, which can provide more meaningful data in comparison to when the line of questioning is strict [33].

All three of the interviews were conducted using Microsoft Teams while being recorded and auto-transcribed using the same tool. The interviews were then manually transcribed as needed before the data analysis. The interviewees could decline being recorded. All interviews lasted between one hour and one hour and thirty minutes. The interview guide for the first set of interviews can be found in Appendix A.

In addition to the interview guide, the artifact version of Cycle I was presented for further explanations and discussions of the suggested solutions with the aim of inspiring the interviewees to come up with additional ideas as part of Cycle II's problem investigation.

The third round of interviews was part of Cycle II's design validation phase. Three interviews were conducted, all with experts who are employees of the telecommunications company. The first two interviews were conducted via Microsoft Teams while the third one was held in-person at the interviewee's company premises. The interview guide for the first set of interviews can be found in Appendix B.

Table 3.2 includes general background information of the interviewees. They are from an industrial or academic background, yet all interviewees are experts in requirements engineering and management. Specific roles of participants are not included as part of background information for maintaining anonymity. The included information is indicative of whether they have an academic or industrial background, their experience, and which cycle they were a part of. Specific roles of participants are not included for maintaining anonymity of the interviewees.

Table 3.2: A list of interviewees along with their role, their experience working with requirements and the cycle in which their interview was part of.

Interviewee	Background	Experience (Years)	Cycle
1	Industry	6+	I
2	Academic	6+	I
3	Academic + Industry	4-6	I
4	Industry	1-2	II
5	Industry	2-4	II
6	Industry	1-2	II

Direct quotes from interviewees are included in the findings section, some of them have been altered slightly, i.e. by removing filling words for better readability and replacing words with clarifying words in brackets to provide the context of their statements.

3.4.2 Workshops

A workshop is an event where a group of people comes together to share and acquire knowledge, solve problems or participate in innovation regarding a specified domain-specific problem [34]. Workshops as a research method usually satisfy the expectations of both their conductors and their participants. The conductors of a workshop have the goal of producing reliable data about the domain-specific subject, while the participants' goals relate to satisfying a personal interest within the domain [34].

Three workshops were a part of the data collection of this research, two of those were held by the Software Center while the third one was an independent workshop conducted by the researchers.

The first workshop was held in October 2021 and consisted of researchers and participants from software centered companies within the Gothenburg area. Large-scale agile was discussed in the context of managing requirements, as well as code and tests, and provided insights into current challenges within the domain. The content of this workshop was used as inspiration and reference for the scoping of the thesis.

The second workshop was conducted, as evaluation of Cycle II, at the end of April 2022 and included a similar set of participants as the first workshop; a mix of participants with either an academic or industrial background. During this workshop, the concept of this research was demonstrated and most recent findings were shared. A voting session was performed during the workshop, where participants were shown a list of timings of feedback along with the corresponding feedback types, as per the state of the findings at the time suggested to be the most useful based on findings of the previous cycles. Participants voted for every type they considered appropriate at the corresponding timing, in order to evaluate which of the proposed solutions they considered beneficial for supporting the automated feedback problems of agile development (RQ2).

A final validation workshop was conducted at the end of Cycle III, late May 2022, to evaluate the final version of the artifact, i.e. whether the proposed solutions are beneficial for supporting the automated feedback problems of agile development as

well as how valuable those solutions are (RQ3) and conclude the research. The purpose of the workshop was to evaluate the value of the findings of the thesis work, in addition to get expert inputs and opinions on any missing aspects of the research.

The workshop was conducted virtually using Microsoft Teams, for the reason of acquiring more participants. It was believed that online participation would make it easier and more comfortable for participants to show up, as well as making interactive aspects easier since they required access to an internet-connected computer.

3.4.3 Survey

An online survey was used as the solution validation of Cycle III, with the purpose of gathering quantitative data. The focus of the survey was to recognize, for each point in time in the development process, which role should receive each type of feedback. The reason for conducting the survey was the need to connect the findings from previous cycles in order to be able to illustrate them clearly.

The benefits of using an online survey at this point were that it would provide quicker data compilation as well as being a quicker solution than doing an in-person survey or interview. However, the main drawback of doing an online survey is that the response rate is generally lower than for in-person surveys [35].

The survey was created using *SoSciSurvey*[2] based on consultation with supervisor and industry contact making sure to use an acceptable tool for the employees of the telecommunications company. The survey is included in Appendix C.

Each of the main survey questions were presented as a multiple choice matrix with types of feedback as rows and receivers of feedback as columns. The survey design was selected with the purpose of getting as much relevant information as possible without making the survey too complicated. This was decided considering the risk of getting too few answers for the relatively short amount of time the survey could be open.

The respondents were asked to select every role that should receive the corresponding type of feedback, first for an individual requirement and then a set of requirements. The survey included a total of seventeen questions, including questions regarding the demographics of the respondents. The demographics were collected to help put the answers into context and make sure respondents were relevant to the study. At the end of the survey, respondents were asked for any additional comments, questions, or concerns they wanted to address.

The survey was sent to individuals within the telecommunications company, some who had been interviewed before and some not, as well as to individuals at two automotive companies in the Gothenburg area. Further, the survey was sent to individuals that are part of the Software Center and to interviewees one to three that were part of the interview process as representatives of the open-source version of T-Reqs and academy.

The invite letters encouraged the receivers to forward the survey to other individuals who have an interest in the topic, with the objective of reaching a larger sample. In the end, a total of twenty individuals responded to the survey.

[2]https://www.soscisurvey.de/

4
The Artifact

The artifact combines the results of the study by presenting them as guidelines that suggest how to improve automatic validation of requirements. The guidelines are constructed based on findings from interviews and workshops with practitioners in the field of requirements engineering and include feedback types, timings in a git-based workflow, receivers of feedback, and discussions of the abstraction levels of feedback.

The guidelines are intended to provide instructions for practitioners and researchers within the field of requirements engineering as to how automatic feedback can be incorporated into new or existing requirement tools. These guidelines are intended as suggestions for those requirement specialists to help them determine which feedback to introduce to their own systems based on which problems they aim to solve, their business value, and the implementation effort. The guidelines in their entirety can be found in Appendix A.

4.1 Problems

The guidelines consist of an overview of the problems that arise when simultaneous changes are made to requirements in agile systems. The problems are categorized in the four detected overview categories: Change History, Interdependencies, Language, and Technical Feedback. A fishbone diagram, shown in Figure 4.1, is used to present the overview categories and displays the main problems of requirement verification and validation in agile systems that can be addressed by using automated feedback.

4.2 Solutions

The main body of the artifact are guidelines, representing solutions linked to the respective problems they aim to solve. The solutions are on the form of user stories, listed in table 4.1, which describe how the solutions can resolve the problems they are mapped to. Additionally, each solution has been allocated a business value, from 1 to 3, and a cost of implementation, from 1-4, to determine how valuable the solutions are, and the feasibility of implementing them. A high business value and a low implementation cost indicate a more valuable user story. These values are based on practitioners' ratings of the user stories; how valuable they are to solve the problems they are linked to, and what they estimated their implementation cost to be. The category that the solution presented in each user story belongs to is

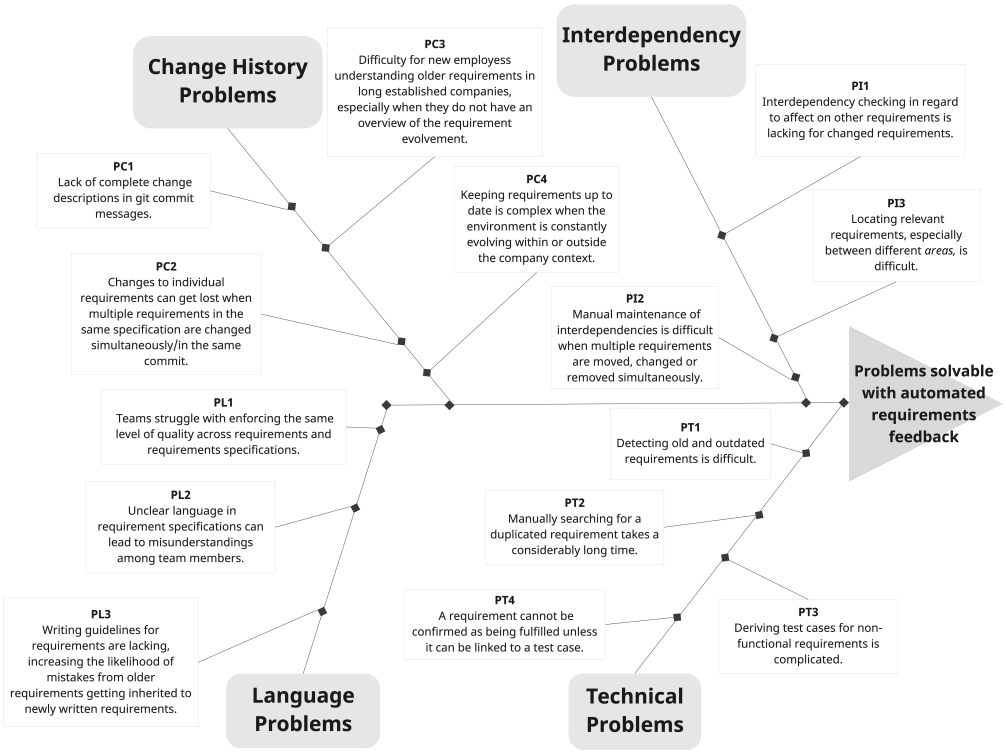


Figure 4.1: Overview displaying the main problems of requirement verification and validation in agile systems that can be addressed using automated feedback, based on problem investigation through interviews and literature.

additionally included in the table. The categories are Interdependencies (I), Change History (C), Language (L), and Technical feedback (T). An example of a user story is shown in Figure 4.2, and the user stories are structured in the following way:

As a <user> I want to <solution> so that <solved problem>.

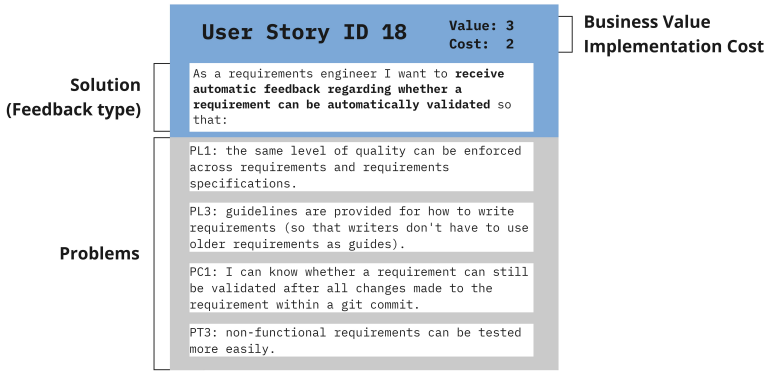


Figure 4.2: An example showing how user stories are formulated and displayed in the guidelines.

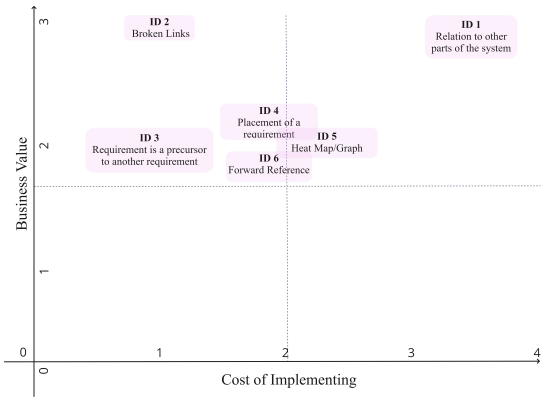
Each solution has suggestions of timings in the workflow where they can be applied and to whom to deliver the feedback at each timing. The timings were derived from interviews with practitioners and requirements experts, while the receivers of each type of feedback at each point in time were based on the survey results.

The guidelines provide suggestions in the form of a mapping between each timing that the feedback should be given and the receivers that should receive it at that particular timing.

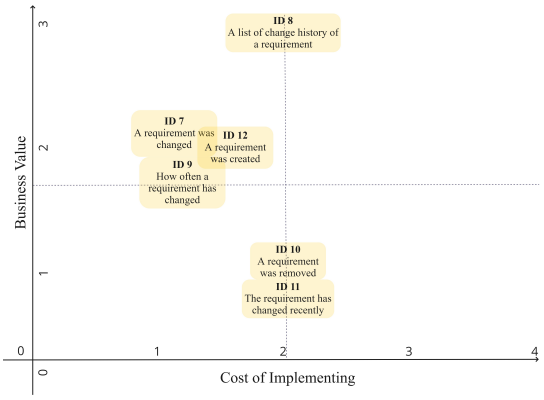
Finally, the guidelines include proposed suggestions for abstraction levels based on the results. The proposed abstraction levels are: error, warning, information and scoring system. Information on the abstraction level that each specific receiver should get at the respective timing needs further investigation.

4.3 Value of Solutions

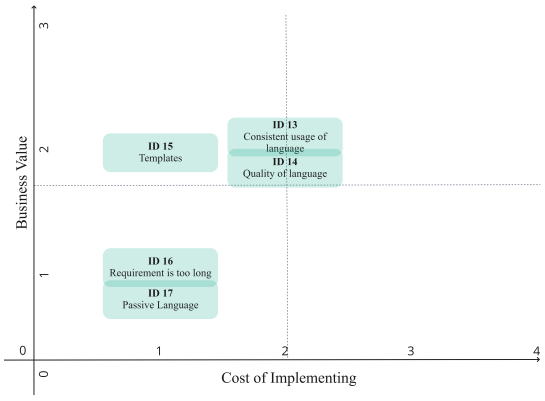
The guidelines display overview matrices, as shown in Figure 4.3, of the business value against the cost of implementation for each of the categories of feedback types; interdependencies, change history, language, and technical feedback. The cost and value of the user stories were decided by requirements experts during a workshop and are also displayed in the heading of each user story. Larger figures displaying the matrices can be found in Appendix A.



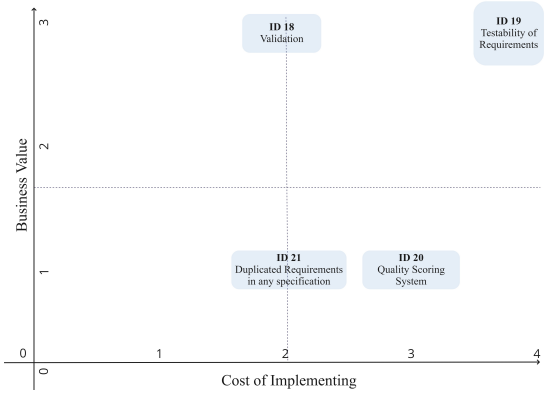
(a) Cost of implementation and business value matrix for **interdependency** user stories.



(b) Cost of implementation and business value matrix for **change history** user stories.



(c) Cost of implementation and business value matrix for **language** user stories.



(d) Cost of implementation and business value matrix for **technical** feedback user stories.

Figure 4.3: Matrices that show the business value in relation to the cost of implementing user stories, as rated by requirements experts. Larger figures displaying the matrices can be found in Appendix A.

Table 4.1: Business value and implementation cost of each of the proposed solutions, the feedback types, based on rating by practitioners during validation workshop. The ID represents the id of the feedback type the corresponding user story refers too. The category displays if the feedback type is categorized as Interdependencies (I), Change History (C), Language (L) or Technical Feedback (T).

ID	Category	User Story	Value (1-3)	Cost (1-4)
2	I	As a requirements engineer I want to automatically get notified of broken links.	3	1
8	C	As a requirements engineer I want to automatically get a listing of all the changes that have been made to a requirement.	3	2
18	T	As a requirements engineer I want to receive automatic feedback regarding whether a requirement can be automatically validated.	3	2
1	I	As a requirements engineer I want to be able to see relations to other parts of the system.	3	3.5
19	T	As a requirements engineer I want to receive automatic feedback on whether a requirement is testable.	3	4
15	L	As a requirements engineer I want to receive automatic feedback on requirement's compliance to a corresponding template.	2	1
9	C	As a requirements engineer I want to automatically see the number of times a requirement has been changed.	2	1
3	I	As a requirements engineer I want to automatically be told that a requirement is a precursor to another requirement.	2	1
7	C	As a requirements engineer I want to be automatically told that my requirement has changed recently.	2	1
12	C	As a requirements engineer I want to automatically get an indication that a requirement has been created within a certain specification.	2	1
13	I	As a requirements engineer I want to receive automatic feedback on the consistency of language use within a requirement.	2	2
14	L	As a requirements engineer I want to receive automatic feedback on the quality of language within a requirement.	2	2
6	I	As a requirements engineer I want to be notified when a functionality is used in my requirement before it is defined.	2	2
4	I	As a requirements engineer I want to automatically know the placement of a requirement.	2	2
5	I	As a requirements engineer I want to be able to see a heat map / graph displaying all connections between the relevant requirement and other requirements.	2	3.5
16	L	As a requirements engineer I want to receive automatic feedback if a requirement is too long.	1	1
17	L	As a requirements engineer I want to receive automatic feedback about passive language within a requirement.	1	1
10	C	As a requirements engineer I want to automatically get an indication that a requirement has been removed from a specification.	1	2
11	C	As a requirements engineer I want to automatically get an indication that a requirement in the respective specification has been changed.	1	2
21	T	As a requirements engineer I want to receive automatic feedback about duplicated requirements in any specification.	1	2
20	T	As a requirements engineer I want to receive automatic feedback on the quality score of a requirement.	1	3

5
Findings

The findings of the study are a combination of problems identified within requirements verification and validation, solutions to the identified problems and finally an evaluation of how valuable the suggested solutions are in terms of solving the problems. Each of the study's research questions has a dedicated section where they are answered in detail. The artifact in chapter 4 builds on the answers to the research questions.

5.1 RQ1 (Problem)

What are the main problems of requirements verification and validation in agile systems that can be addressed using automated feedback?

As defined by Gause and Weinberg [36], a requirements engineering problem is the difference between what is desired and how things are perceived, i.e. the discrepancy between how users expect or *desire* the system to work and how they *perceive* it to work.

Considering that the goal of the study was to discover which types of automated requirements feedback is the most beneficial when integrating multiple parallel changes to requirements in agile environments, the study focused on identifying the problems perceived by practitioners who work with requirements as well as getting input from researchers involved in requirements engineering research.

The data collected throughout the research process suggests that multiple problems exist when it comes to verifying and validating requirements in agile systems.

5.1.1 Key Issues

The key issues of requirements management, considering that they should be verified and validated, were identified through a literature review during Cycle I. The issues can all be related back to the fact that being flexible to changes while planning continuously is a crucial part of agile processes [4] and the management of requirements must therefore also be a flexible process in order to maintain agility among cross-functional teams [5]. Requirements not being rigid from the start in agile software development poses a challenge for continuous requirements management as the requirements may change over time [7]. This call for flexible planning has resulted in obstacles within the requirements verification and validation process.

Teams commonly being distributed and agile makes it difficult to collaborate and

coordinate requirements within critical systems since such systems require thorough documentation [15]. Meanwhile, information regarding changes should be communicated with minimal delay [5]. Traditional ways of manually reviewing requirements can take a long time based on agile standards, so getting real-time feedback on changes would be a great improvement to this task [14].

Requirements validation is one of the biggest challenges of traditional Requirements Engineering, especially when it comes to traceability between tests and requirements as well as relations between information items [4]. Thus, there is a need for continuous validation of requirements of a product within agile contexts where customer needs may change overtime [4].

5.1.2 Timings and Receivers

Automated requirements tools seem to be the most beneficial when they are combined with manual inspection in order to avoid ambiguity-related faults [25]. Before identifying which types of feedback can be used to solve issues within requirements verification and validation, it was observed in multiple interviews that the value of giving feedback relies heavily on it being given at appropriate times. As an example, one interviewee noted the following:

> *"If you're just ignoring the recommendations all the time then the feedback is not useful."*
>
> *-Interviewee 1*

Three conclusions were drawn from interviewees' concerns regarding the usefulness of requirements feedback. First, knowing which feedback to give at which time is complicated since different feedback types are appropriate at different times. Second, bad feedback frequency causes people to ignore it. Third, too much unnecessary feedback causes people to ignore it. Thus, studying the times at which different feedback should be given and to whom became an apparent problem that had to be considered when the problems were analysed and solutions to them suggested with receivers of feedback in mind.

5.1.3 The Problems

More specific issues relating to requirements feedback were identified as problems in subsequent cycles, during discussions with interviewees who study, create or by any other means work with requirements. Those issues were grouped into four categories: Change History, Interdependencies, Language and Technical Feedback.

5.1.3.1 Change History

When multiple teams are working in parallel on the same requirements, they need to have the ability to report changes to them without blocking each other [6]. Moreover, data collected through interviews with requirement experts suggested that the ability to explore the history of changes made to requirements is vital to the requirements verification and validation process, however the existing tools could be improved upon to provide better support for requirements. The problems of requirements verification and validation that can be connected to the change history

of requirements are listed below.

PC1: Lack of complete change descriptions in git commit messages.
It was reported that git commit messages do not always capture everything that was changed in a requirement. Interviewee 5 disclosed the following during an interview:

> *"Quite often, the changes that we have made to the requirements may come in a larger change, so it could [happen] that someone is overhauling this document or adding a lot of functionality. Then it might be that the commit message isn't great, it isn't really capturing everything that is included and why the change actually is happening. So getting that context in is important, but a bit lost today."*
>
> *-Interviewee 5*

Thus, it is problematic not to be able to have a clear view of what has been changed in a requirement since that means that information about what and why things were changed can be lost.

PC2: Changes to individual requirements can get lost when multiple requirements in the same specification are changed simultaneously/in the same commit.
Interviews revealed that not being able to view changes made to individual requirements, especially in T-Reqs, is problematic. An interviewee noted that:

> *"If you look at our requirement specification, how it works today, [viewing changes to an individual requirement] is not really possible. You can see it on a document level, but I think it is a bit difficult to see the change history of a single requirement with how T-Reqs looks today. Which I think that can be a bit problematic. Even though there is a a change history, people will not really see it. You don't want to go through 20 document changes to see what has changed. You want to find something just about the requirement that you are interested in."*
>
> *-Interviewee 6*

The takeaway is that it is currently difficult to see the changes made to an individual requirement, since it is hidden behind the change history of all requirements in the document.

PC3: Difficulty for new employees understanding older requirements in long established companies, especially when they do not have an overview of the requirement evolution.
Long established companies can have very old products with old requirements that new employees struggle with understanding. An interviewee supported this claim with the following statement:

> *"One of the things I struggled with the most when I started was that we had all those requirements, I mean our product is super old. We have been working in the same code-base for twenty or thirty years so if you don't have a change history then it is very hard to know what is going on."*
>
> *-Interviewee 5*

This essentially means that new employees often struggle with understanding existing requirements, and thus have troubles understanding the system, when they can not see how the requirements have evolved.

PC4: Keeping requirements up to date is complex when the environment is constantly evolving within or outside the company context.
Keeping everything up to date is problematic when technology is constantly evolving and the requirements within the company or from the customers can change. An interviewee noted the difficulty of keeping everything up to date around them:

> *"Things change in the world around us. Within [the company] and outside. So, keeping all of this up to date is tricky."*
>
> *-Interviewee 5*

This can stem from a variety of reasons, such as societal changes, technology improvements and customer wants. Therefore it is problematic when requirements are not kept up to date.

5.1.3.2 Interdependencies

As explained by Carlshamre et al. [37], interdependencies stand for interactions between requirements, as requirements most often relate to each other in one way or another. Further, they state that interdependencies can make it difficult or seemingly impossible to schedule the order of which requirements should be implemented.

From the validation process of Cycle I, it became clear that interdependencies are not documented to a great extent at the telecommunications company in question and that their requirement specifications might benefit from feedback regarding interdependencies.

PI1: Interdependency checking, in regard to affect on other requirements, is lacking for changed requirements. During the problem investigation interview of Cycle I, an interviewee who has a lot of experience with managing requirements and with T-Reqs said that:

> *"[Interdependencies are] something that we are lacking [at my place of work]."*
>
> *-Interviewee 1*

Another interviewee supported problem of lack of interdependency checking, stating:

> *"I think that this has happened to me where I'm trying to link to other stuff in other documents and the system just doesn't support that right now."*
>
> *-Interviewee 4*

Interviewees indicated that interdependency checking needs to be improved at their company, which they also revealed has a mix of old and new products as well as working with a variety of tools, some old and legacy.

PI2: Manual maintenance of interdependencies is difficult when multiple requirements are moved, changed or removed simultaneously.
Results of interviews indicated that moving, changing and removing requirements can be problematic as it will break relations to other requirements unless there is

adequate interdependency maintenance involved. An example mentioned by interviewee 6 was that:

> *"One thing that we have an issue with is that we have a lot of documents and you often want to have links between them."*
>
> *-Interviewee 6*

They also indicated that their company has the issue that the links become broken if someone moves a document or makes changes.

PI3: Locating relevant requirements, especially between different areas of work, is difficult.
Maintaining linked requirements was reported as a difficult task for individuals who work with requirements, especially that they struggle with locating relevant requirements that belong to different areas of work. When discussing interdependencies and linked requirements, an interviewee noted the following:

> *"It is quite painful to figure out where the requirement is now."*
>
> *-Interviewee 6*

In the same discussion, they mentioned that having an overview of interdependencies is "extremely important", noting that it is one of the recurring issues that their company struggles with since there is not enough automated control of linked requirements.

5.1.3.3 Language

The reviewing process of requirements depends substantially on the requirements being understandable. Flaws in the way requirements are written can lead to misinterpretation, posing a huge risk as it can lead to rework, which can be expensive, or even cause safety-critical errors [14]. Interviewees mentioned the following issues when it comes to the language of requirements:

PL1: Teams struggle with enforcing the same level of quality across requirements and requirements specifications.
When there are no clear guidelines as to how requirements should be written, teams and team members can end up using different writing styles and their understanding may differ when it comes to what a high quality requirement looks like. In fact, this can lead to rework as interviewees reported that they take it upon themselves to fix each other's mistakes, or they have to make fixes when the requirements are difficult to understand or are simply wrong because of how they are written.

PL2: Unclear language in requirement specifications can lead to misunderstandings among team members.
The importance of clarity when requirements are written were directly mentioned during the interviews as the following statement exhibits:

> *"A lot of the times, unclear language can lead to misunderstandings."*
>
> *-Interviewee 4*

This issue relates to the previous language problem of teams struggling with upholding the same level of quality within requirements, since misunderstandings often

stem from differences in the language of requirements.

PL3: Writing guidelines for requirements are lacking, increasing the likelihood of mistakes from older requirements getting inherited to newly written requirements.
A recurring theme identified during the interviews was that individuals who are responsible with the writing of requirements struggle with knowing how to write the requirements. Thus, they resort to looking up existing requirements and use them as templates for new requirements. However, this can mean that old requirements that have not been updated in a long time are not on par with the current writing standards, leading to mistakes from older requirements being passed on to new requirements.

5.1.3.4 Technical Feedback

Remaining problems were positioned in the category of technical feedback. They relate to the validation and testability of requirements, as well as issues with handling duplicated requirements.

PT1: Detecting old and outdated requirements is difficult.
Interviewees mentioned technical debt as one of the biggest challenges that practitioners encounter when working with requirements. For example:

> *"There's a lot of really old requirements, so I wish there were more systems to detect that and more resources spent trying to fix it."*
>
> *-Interviewee 6*

When requirements as well as ways of working change over time, outdated requirements can become problematic as they might still have links to other requirements and add noise to the requirement specifications.

PT2: Manually searching for a duplicated requirement takes a considerably long time.
Although it depends on the size of the requirements specification under investigation and how many different use cases it involves, manually searching for duplicated requirements can be a slow process. In fact, an interviewee who often has the job of searching for duplicated requirements reported that the investigation phase of the most recent feature they worked on, which is conducted before writing the requirements and involves searching for duplicates beforehand, took them about three days.

PT3: Deriving test cases for non-functional requirements is complicated.
When discussing the validation process of requirements, interviewee 6 mentioned that they believe non-functional requirements are more difficult to test in comparison to functional requirements. This means that testing the system design is more difficult than testing code functions and behaviour, since some of the words used when describing non-functional requirements can be hard to test.

PT4: A requirement cannot be confirmed as being fulfilled unless it can be linked to a test case.
One of the examples mentioned by Torkar et al. [18] of how it can be validated that a requirement has been fulfilled is by checking whether it can be traced to a test case, thus the test case's existance indicates that the software fulfills the requirements. Similarly, since the main way of knowing if something is good or not is by somehow testing it, an interviewee's quote rings true:

> *"You can't really say that you have fulfilled a requirement if you don't have a test for it."*
>
> *-Interviewee 6*

Essentially, it is problematic when requirements do not have clear links to test cases. The tests might exist, but if links are missing, it is difficult to have a clear image of which requirements have already been tested and which have not been tested. Further, there are difficulties identifying whether the tests have passed or not.

5.2 RQ2 (Solution Candidates)

What are potential solutions that can be used to incorporate continuously automated feedback to improve the quality of requirements in agile systems that have continuous parallel changes?

Getting automatic real-time feedback on changes would be a great improvement to the review process of requirements, instead of work having to be done manually [14]. A tool that can incorporate real-time feedback into the requirements review process is the text-based requirements management tool T-Reqs, which can be used to incorporate agile conventions into large-scale projects with cross-functional teams [6]. T-Reqs has proven useful for requirements management in large-scale agile companies with a git-based infrastructure [13].

The potential solutions that can be used to incorporate continuously automated feedback to improve the quality of requirements in agile systems that have continuous parallel changes are listed in the following section. They are split into four categories respective of the problems discussed in RQ1, based on the type of issue that they aim to solve:

Change History
Information related to changes made to a single requirement or a collection of requirements.

Interdependencies
The relations between requirements and/or requirements specifications, as well as between requirements and other artifacts.

Language
The consistency, clarity and quality of how requirements are written and how

these factors can be improved.

Technical Feedback
Related to the validation and testability of requirements, as well as issues with handling duplicated requirements.

The results suggested that two groups of feedback should be considered for giving feedback on requirements, as the relevance of feedback differs depending on the surrounding context. Firstly, feedback can be relevant for a single requirement at a time, which puts it in the category of *feedback for individual requirements*. Secondly, feedback can be relevant for multiple requirements at a time, putting them in the category of *sets of requirements*. This indicates that feedback should occasionally be given for an entire set of requirements at the same time.

Individual Requirements
Automated feedback is being given within the scope of a single requirement.

Sets of Requirements
Automated feedback is given within the scope of a collection of requirements, such as a requirements specification that includes multiple individual requirements. It can additionally be any other set of requirements that are grouped together, i.e. are a part of the same specification, relate to the same functionality or are of the same type.

5.2.1 Interdependencies

Five solutions that give support to solving problems regarding relations between requirements were categorized as interdependency solutions. They involve interactions between all parts of the system, both within the same area of work and between different areas.

The findings show that the interdepenency feedback solutions have the potential to help solve nine of the problems of requirements verification and validation, of any problem category, as shown in Figure 5.1.

ID 1: Relation to other parts of the system
Shows relations to other parts of the system, e.g. to other requirements, requirements specifications, critical artifacts or requirements, or to relevant test cases. Also relates to connections to specific customer information, such as who the customer is and who is their contact person. Other examples include whether a requirement has been removed, but the test case remains, or if test cases are valid. Relation to other requirements also refers to where requirements were tested and how. An interviewee noted that:

> *"It could be useful to get to know if your requirement is connecting to multiple requirements in another specification, then maybe it should be there instead of in your [specification]."*
>
> *-Interviewee 5*

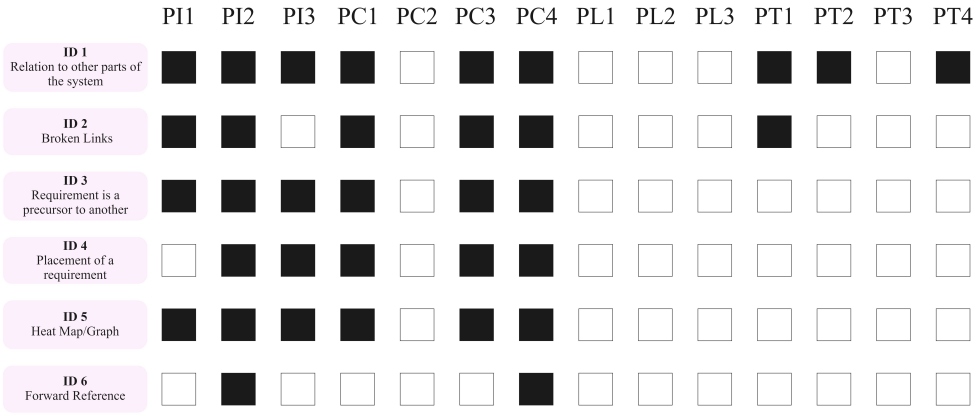


Figure 5.1: Correlation matrix that shows the problems of requirements verification and validation on the horizontal axis, listed by their ID, and whether they can be solved by interdependency solutions that are on the vertical axis.

Interviewees agreed that relations between requirements as well as artifacts are difficult to maintain and understand when the documentation is not adequate. They all spoke positively of the idea to have automated control of the relations between requirements and any other system parts.

ID 2: Broken links
Indicates that a requirement's change will or has broken a link between any two or more artifacts within the system, e.g. the requirement that is being linked to does not exist or the requirement being removed is linked to another requirement.

Breaking links was reported as being problematic, and contrary to how code simply does not compile if there are errors, a problematic requirement can be silently pushed to the repository. Therefore, there should be measures in place that prevent broken links from being pushed into the code-base.

ID 3: Requirement is a precursor to another requirement
Indicates that a certain requirement has to be implemented before another requirement. Interviewee 5 explained the need for the feedback type:

> *"It might be that I have a requirement that I say: for this to work, I need this other functionality to work as well. It is a precursor, and then I may want to know if those requirements have been tweaked or removed or, yea thinks like that."*
>
> *-Interviewee 5*

Knowing whether a requirement is a precursor to another requirement or requirements can help practitioners make informed decisions when changing, moving or removing requirements, as well as helping them understand how the system works in general.

ID 4: Placement of requirements
Indication of the placement of requirements. Can be within a certain specification,

in relation to connected requirements or in relation to workflows within your company. Practitioners indicated that requirements specifications are usually divided by areas of work or by feature, so the ability to easily locate requirements would help solve problems related to requirements being placed in the wrong specifications.

ID 5: Heat Map/Graph
A visual representation of all connected requirements. Shows, for example, all connected requirements to the relevant requirement and/or all connected artifacts. Additionally, a useful way to identify dead requirements or dead specifications, i.e. outdated requirements or specifications that are no longer in use. The purpose of a heat map or graph was explained in an interview:

> *"It could be some sort of sanity check on 'Perhaps my area should not be dependent on from some other area in order to have the correct system architecture'. I don't want to have a requirement spread out into too many different specifications, and which way things are linked could be important. If it starts looking too much like a web then perhaps I should consider moving my requirements around to make for a clearer specification that could be the use case I see for such a graph."*
>
> *-Interviewee 5*

Other practitioners also thought of heat maps as a good way to represent relations between requirements. Interviewee 6 mentioned that such a heat map would be "fantastic" since it could help discover dead requirements, and practitioners were particularly interested in seeing interdependencies in a heat map for sets of requirements.

ID 6: Forward Reference
Definitions are in the wrong order in a way that a functionality is used in a different requirement before it, itself, is defined [19]. This means that an element is used in another requirement before it is defined itself. Interviewees indicated that forward referencing is wrong in general, so they would like to get an indication that they're making a forward reference or that a forward reference exists, so that they can avoid making that mistake.

5.2.2 Change History

Knauss et al. [20] discusses that in larger organizations, where the work environment is collaborative, there is the need for teams to have knowledge of who is working on related tasks, has already finished them or is waiting for input as support for quality management for projects.

Viewing changes made to code or requirements is an existing feature in version control systems such as GitLab [38] and Gerrit [39], where inspiration for this category was drawn from.

Therefore, a prominent categorization identified during the interviews relates to the change history of requirements. Those are solution candidates of automated feedback that support solving the common problems related to change history of requirements. The change history feedback solutions are intended to help solve six

of the problems of requirements verification and validation, of any problem category, as shown in Figure 5.2.

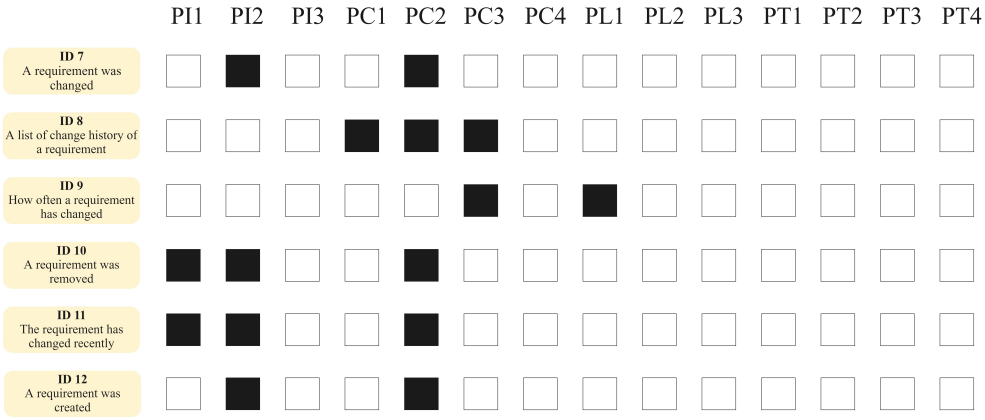


Figure 5.2: Correlation matrix that shows the problems of requirements verification and validation on the horizontal axis, listed by their ID, and whether they can be solved by change history solutions that are on the vertical axis.

ID 7: A requirement was changed

Indicates whether some change has been made to a requirement since the last time the person viewed it. Also indicates when it was changed.

ID 8: A list of the change history of a requirement

A list showing the entire change history of a single requirement. The list can then be used to access a summary of the nature of those changes. This solution displays an overview of the information other change history solutions provide.

ID 9: How often a requirement has changed

A numerical value indicating how often a requirement has been changed since creation. An interviewee discussed the value of knowing how often a requirement has changed:

> *"[I] think it would be quite simple to calculate such a statistic. Then getting that statistic could be useful for someone wanting to look into any requirements that are bit unstable, and should we take extra control of it? Have someone look into it a bit more? So I think that you can find quite a few use cases where knowing how often the requirement has been changed would be be useful."*
>
> *-Interviewee 5*

This can help with locating dead requirements that have not changed in a very long time so that they can be inspected, or alternatively to see whether requirements are changing a lot and are therefore unstable.

ID 10: A requirement was removed

An indication that a requirement has been removed along with information of the change e.g. who removed the requirement and why. A requirement being removed is

similar to changing it, since both indicate changes to the requirements specification, but is more likely to break links as linked requirements will then point to a non-existing requirement.

ID 11: The requirement has changed recently

An indication that a requirement has been changed in any way, recently. The requirements engineer should be able to see *if* something has changed within a requirement, as well as seeing *what* has changed. Interviewees mentioned that it would be very useful to see that a requirement has changed, especially when requirements are changing iteratively.

ID 12: A requirement was created

An indication that a new requirement has been created within a relevant specification. As part of the change history of requirement specifications, newly created requirements should be indicated since they might otherwise get lost, especially when multiple changes have been made to the same specification. An interviewee noted that:

> *"You can see [changes to requirements] on a document level, but I think it is a bit difficult to see the change history of a single requirement."*
>
> *-Interviewee 5*

Since a requirement cannot be changed until it has been created, its creation can be considered a change since it went from being non-existing to existing. However, clarifying that something has been added to a requirement specification was considered important by practitioners, especially since new requirements should also be reviewed and linked to test cases. Note that this feedback type was added during Cycle III and thus was not included in the survey that connected the solution to timings and receivers.

5.2.3 Language

Four solutions were identified as having the aim of solving problems relating to the language of requirements. They should encourage requirement engineers to write requirements of higher quality, with more consistency and fewer misunderstandings among relevant parties.

The language feedback solutions are intended to help solve seven of the problems of requirements verification and validation, of any problem category, as shown in Figure 5.3.

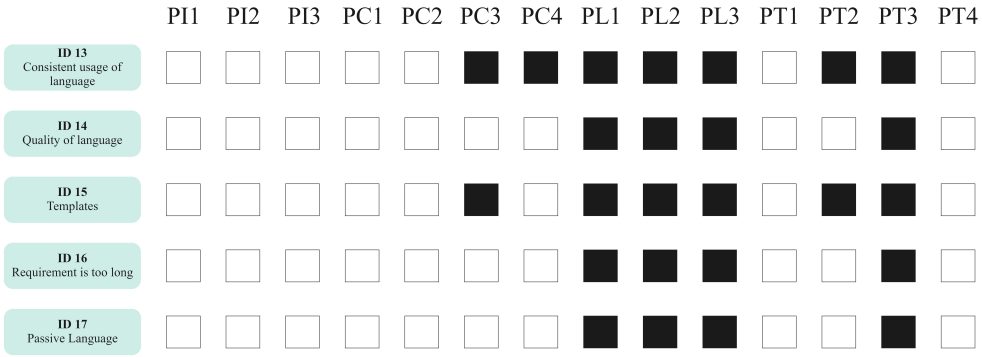


Figure 5.3: Correlation matrix that shows the problems of requirements verification and validation on the horizontal axis, listed by their ID, and whether they can be solved by language solutions that are on the vertical axis.

ID 13: Consistent usage of language
Alert if usage of language might not be consistent with language of other requirements within the domain. Includes suggestions of domain specific word usage or acronym definitions. An interviewee supported the need for feedback on consistent language, stating that:

> *"It could also be that there are specific words that have a special meaning in the requirement space, like the words may or shall, they are important to use in the correct way, and you might want to have the same definition of what those words mean. I have seen us add 'this requirement specification is according to this standard' but it may be better then to say that all the requirements here should follow the same standard and if you hover over the word shall, there will be an explanation of shall, there will be an explanation of that shall is actually used in this way."*
>
> *-Interviewee 5*

Interviews with practitioners further revealed that there are expectations of a mutual understanding of how requirements should be written language-wise, yet they reported that this is often not the case. Skill levels and preferences of those working on requirements vary which leads to inconsistencies in how requirements are written.

ID 14: Quality of language
Automatic feedback of language quality (e.g. grammar, clearness and complexity) while writing the requirement. Syntactical rules are important when striving to

achieve good quality in requirements [20]. Ormandjieva et al. state that their hypothesis is that "the root cause of errors being introduced into the requirements (and the consequent reduction in quality) is ambiguity in the text" [19, p. 2]. Practitioners echoed those claims during interviews, as they struggle with enforcing the same quality across the board. They also reported having struggled with understanding low-quality requirements.

ID 15: Templates
Adherence of requirements or specifications to pre-determined templates of the corresponding type was mentioned during interviews as a helpful solution towards maintaining good writing standards.

> *"The advantage of [templates] is that they give you a fixed sentence structure. The sentence is the system shell, do whatever. And in some cases, these can actually be used to automate things."*
>
> *-Interviewee 2*

For individual requirements they can be on the form of suggestions and examples of how to formulate a good requirement, while for sets of requirements the templates can be suggestions of requirement specification structure or placement of individual requirements in relation to other requirements in the same specification.

ID 16: Requirement is too long
Indication that requirement is too long. Ormandijeva et al. [19] suggest that multiple factors can ease understanding of documents. Parts of those factors are sentence length and unnecessary words or phrases. Interviewees agreed that requirements should be as simple as possible and should be broken down if they are too long. The length of requirements can suggest different problems within them, for example not being atomic, as mentioned by interviewee 3:

> *"I would check if requirement is atomic and for example words like 'and', 'or' and so on, they indicate that a requirement might not be atomic and then information could be printed out to the person who wrote their requirement."*
>
> *-Interviewee 3*

For those reasons, the feedback includes suggestions of how the requirement could be split, of redundant words, or usage of connective words indicting more than on feature description in the requirement.

ID 17: Passive Language
Informs if usage of passive language is detected and proposes better language usage without it, for example by changing the sentence or removing passive words and phrases. When suggesting a benchmark for the quality of textual requirements, Ormandjieva et al. [19] mentioned, among other things, passiveness of verbs. One interviewee even called passive language a "classic" when discussing language quality attributes of requirements.

5.2.4 Technical Feedback

Technical feedback is a category for the types of feedback that relate to the validation and testability of requirements.

Validation of requirements is considered an important practice to incorporate into the solution in order to support the management of requirements [5]. The technical feedback solutions are intended to support the verification and validation processes of requirements via automation, testing, quality inspection and duplicate search.

The technical feedback solutions are intended to help solve ten of the problems of requirements verification and validation, of any problem category, as shown in Figure 5.4.

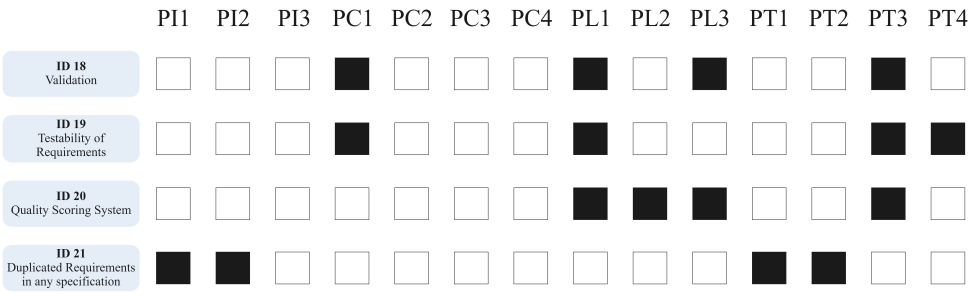


Figure 5.4: Correlation matrix that shows the problems of requirements verification and validation on the horizontal axis, listed by their ID, and whether they can be solved by technical feedback solutions that are on the vertical axis.

ID 18: Validation

Indication of the possibility of automatically validating a requirement. A requirement validation system integrated into an organization's workflow can help with managing system requirements through a version control system, similarly to collaboratively creating and reviewing code [5]. Interviewee 4 noted that:

> *"We already have some validations, for example if the requirement ID is is OK or not."*
>
> *-Interviewee 4*

Building on that idea, checking whether a requirement can be automatically validated can involve checks that it includes the necessary things to assess whether a requirement can be validated. Another part of this idea is checking whether a requirement fulfills the properties required in order to verify it, i.e. that it is described in the correct way.

ID 19: Testability of requirements

Indication of the testability of requirements. Examples of things that could be automated are if there are two or more functionalities incorporated in a single requirement, or if the requirement already exists. Interviewee 3 strongly supported receiving feedback on testability of requirements by stating that:

> *"Of course, if you make a requirement, the requirement must be testable."*
>
> *-Interviewee 3*

Part of knowing whether a requirement is fulfilled is knowing whether it was been tested, as supported during another interview:

> *"You can't really say that you have fulfilled a requirement if you don't have a test for it, in my opinion at least."*
>
> *-Interviewee 6*

Interviewees also discussed that it should be clear whether a requirement is testable and know which form of testing is necessary for that requirement.

ID 20: Quality Scoring System
A value that indicates the quality of a requirement that gives the requirements engineer a suggestion of the quality of a requirement or a requirements specification. Interviewee 4 mentioned such a system and that suggestions of improvements could further be involved:

> *Maybe [as I] also said, like a lot with all the [issues] that can be improved and then a score of the requirement that this requirement is like 90% [in quality] maybe [and then] you can do something else to improve it, something like that.*
>
> *-Interviewee 4*

Interviewees suggested, for example, that such a scoring system can indicate a requirement's impact on the system development:

> *Maybe it can be solved in a different way that doesn't require a change of the hardware platform. So I mean, you see what I mean. So feedback could be if you give a score or like a point, points, in how severe of an impact this requirement would trigger or cause in the system development? A requirement that requires the change of 1000 lines of code is a lot more complicated than a requirement that changes only two lines of the code.*
>
> *-Interviewee 3*

A requirement change involving hundreds of line changes is very likely to have a larger impact on the system development than a one line change would, although there can be no guarantees so other factors should be considered. Other elements of requirements that can be scored are their similarity to other requirements as well as how high quality the language of them is.

ID 21: Duplicated requirements in any specification
Indication whether a requirement has already been defined or that another similar requirement exists. This might be indication of duplication or that the relevant requirement is in the wrong specification if multiple similar requirements are elsewhere.

5.2.5 Receivers and Timings

As described in the problem description (RQ1), identifying the receivers of each type of feedback and when they should receive it is crucial for the feedback to be valuable. The following receivers and timings were identified.

5.2.5.1 Receivers

The receivers of feedback, who were identified as the most relevant to receive feedback automatically when making parallel changes, are listed in the following section.

Author of Requirement
An individual who creates a new requirement or is the writer of the newest change for the corresponding requirement was given the identity of *Author of Requirement*. During interviews and workshops, it was evident that the author of a requirement should be a candidate when it comes to receiving requirements feedback.

It is clear that a review of all written requirements is required for raising their quality, making sure they are clear, correct and understandable. Delivering automatic feedback to the author of requirements ensures the quality of the requirement as it lets them know if there are problems with it, allowing them to correct them before it reaches further in the process. Automatic feedback simplifies the review process and frees up the reviewer's time to validate aspects that can not be automatically validated.

Requirements Responsible/Owner
An individual that is formally responsible for a requirement or a set of requirements was identified as the *Requirements Responsible*. They can also be referred to as the *owner* of the requirement or specification ans was thus identified as the *Requirements Responsible/Owner*. The following interviewee statement explains the main reason why the requirements responsible/owner should get feedback about requirement changes:

> *"If they are not notified then it might be that something is changed and they don't know about it and [then] they can't really control their area in a good way."*
>
> *-Interviewee 5*

During discussions with experts, it became evident that the responsible, or owner of requirements, is highly relevant when it comes to receiving automatic feedback on requirements. The responsible or owner of the requirements needs to be informed of changes to their requirements to keep an overview of their area of work.

Development Team
The group of individuals that create the product from the requirements through development was identified as the *Development Team*.

During interviews with experts, there were discussions regarding whether development teams benefited from getting automatic feedback. As mentioned by interviewee 5, the discussions often implied that they would not benefit from getting the feedback all the time, e.g. that it would be more appropriate for the author of a requirement to process the feedback first:

> *"Maybe [the development team] could wait until I have sort of processed the feedback and you know, then you can hand them a more refined version of it."*
>
> *-Interviewee 5*

Other discussions led to findings that suggested that, in some cases, receiving auto-

matic feedback as a development team would be of advantage for them:

> *"The development team who worked with the requirement should be able to see the history of these feedbacks or changes."*
>
> *-Interviewee 3*

The general conclusion from the discussions is that it can benefit development teams to receive feedback automatically for requirements currently in production, i.e. the requirement they are currently developing. After that, the relevance of the whole development team receiving the feedback decreases.

Test Engineers

The individuals responsible for testing code, making sure the requirements are tested and that the tests pass, were identified as *Test Engineers*. They can be a part of the development teams, depending on the context the teams work in. Regarding test engineers receiving feedback about changed requirements, it was noted that:

> *"Yes, [when creating and changing requirements] it is much more important [than for the development team] both when new requirements are fulfilled, but also when they are removed, especially if it is related to some test case. It could also be useful for testers to know about planned changes, but not as important."*
>
> *-Interviewee 5*

Discussions with interviewees revealed the usefulness of testers receiving automated feedback, especially when requirements are created or when changes are made that can affect the connected test cases. When requirements are changed, the test cases can become obsolete, and when new requirements are added, a corresponding test case needs to be added to test the new functionality as well.

Product Owner

Product owners have the role of setting product goals and making sure that the value created by the development team is as high as possible. This role can vary between different context of companies or teams [40]. Interviewees recognized product owners as being one of the main stakeholders of the product under development and should thus receive feedback on it:

> *"They may not be interacting as much with their requirement directly, but they still care about the area that they are working within."*
>
> *-Interviewee 5*

Essentially, the product owner might not be interested in the exact mechanics of how the requirements are produced, but has the responsibility of ensuring that the final outcome reflects the product goals and should therefore receive feedback related to changes in the product's requirements.

5.2.5.2 Timings

The points in time that were found the most relevant to receive feedback automatically when making parallel changes were collected from examinations of both GitLab [38] and Gerrit [39] workflow. GitLab and Gerrit were examined since they are the ones used in the open-source T-Reqs and within the telecommunications company, respectively. Another way of collecting the timings was through discus-

sions with experts during interviews and workshops.

Writing requirements
According to the data the study examined, feedback should be given in real time when writing requirements, in the graphical user interface or command line interface. Interviewee 2 mentioned getting feedback while writing requirements for the first time, yet also when making changes to requirements:

> *"If you write requirements early, then I think it's good to early on get feedback and otherwise if you're changing requirements. [...] So either when you develop the requirements or when you change them."*
>
> *-Interviewee 2*

Getting feedback while writing requirements can either happen at the beginning of the process, when writing the requirements or when making changes to them later in the process. Getting feedback when writing, and thereby getting feedback early on in the process, was mentioned in multiple discussions with interviewees as a feasible timing.

Committing to the git process
Feedback should be given when a requirement change is committed to the git process. When discussing feedback when committing to the git process, interviewee 4 stated that:

> *"[It] saves time if information is known from the start. So, I would say that entering the commit process, it would be better [than later] because then you can like correct it there. Because maybe those comments can also come later by the reviewers and Gerrit. So, we'll get [to] save that time, probably, if you already know from the start."*
>
> *-Interviewee 4*

In this case, the feedback is given before the requirements are published for others to see. This was discussed as a valuable feedback timing to save reviewers' time since requirement authors would fix their requirements, so reviewers would get a more refined version of the requirements than before they were first committed. That way, reviewers might have a chance to focus their time on giving other kinds of feedback that are not automated.

Pushing changes to code-base or repository
Feedback should be given when requirements have been reviewed and are being pushed to the code-base or repository. At this point the requirements have already been accepted by a reviewer when using Gerrit [38] and are about to be merged. In a GitLab process [38], the changes are being pushed for review. Discussions during interviews and workshops determined some general parts of the git-based processes, such as delivery, as being comparable to pushing to code-base or repository. Thus, such timings are included here.

Reviewing someone else's changes
According to the participants of the interviews and workshops, feedback should be given when manually reviewing changes that someone else has committed or pushed

to the relevant process, i.e. reviewing a merge request in GitLab [38] or doing a code review in Gerrit [39].

During automatic maintenance checks
Automatic maintenance checks of requirements and requirements specifications was an idea that came up during an interview and was approved by participants in later workshops. The idea was that a maintenance program would produce automated feedback on schedule, at certain intervals, or triggered on demand. As an example, interviewee 6 mentioned running background jobs with the purpose of detecting poor requirements:

> *"I think maybe [I'd like to get feedback on] stuff that I'm not working with actively. I think it would be nice if there were jobs that would go through existing requirements and try to flag poorly written ones."*
>
> *-Interviewee 6*

Examples of other automatic maintenance checks mentioned by interviewees were for example to find low quality requirements or to locate areas of improvement.

Any time
The experts interviewed as well as those who participated in the workshops showed a clear desire to be able to access feedback at any time, i.e. on demand during or after the general work process. The feedback is generated on demand. Accessing feedback any time was supported both during workshops and interviews, for example:

> *"So, I think this automated feedback should be available at any point in the requirement life cycle. If there's such a thing as a requirement life cycle."*
>
> *-Interviewee 3*

This means that, essentially, feedback should be accessible at any time, from when a requirement has been created and until it is no longer existing. Interviewees mentioned that it is useful to have access to feedback any time there is a need to revisit the requirements, such as when researching requirements or studying the requirements for new features.

5.2.5.3 Connecting Timings and Receivers

As mentioned in RQ1, feedback is not likely to be of value if the timing of delivering it and its receiver of the feedback are not relevant. The following findings connect each timing and receiver solution to each of the feedback types discussed in sections 5.2.1 to 5.2.4. The connections were represented in heat maps presented in Figure 5.5, 5.6 and 5.7, which display receiver and timing information for each of the solutions, the feedback types, except for user story 12 which was added after the survey was conducted. Each row represents a timing while each column represents a receiver of feedback. The timings for each type of feedback are based on evaluation conducted in the evaluation workshop of Cycle II. The values of the table represent the percentage of participants in the conducted survey, that selected the corresponding receiver at the corresponding timing for the feedback type in question. The darker the color displayed in the table, the more respondents chose between a timing and a receiver

for the relevant feedback type.

Analysis of the heat maps was done using 50% as a threshold for deciding which receivers are recommended for each timing. Specifically, if the value between a timing and a receiver is higher than 50%, those are determined a valuable combination and are included as part of the guidelines. This means that, at least eleven out of twenty practitioners agreed that these individuals should receive the respective feedback, at the corresponding timing.

The analysis of the heat maps showed that for most feedback types it is the author of requirements that should be the receiver of feedback, at most points of time in the workflow. The second most relevant receiver is the requirements responsible/owner. Other receivers did not get survey responses above 50%.

Notable results are that for the timing *During Maintenance Checks*, the requirements responsible/owner is the most important receiver of feedback in all cases. Furthermore, the requirements responsible/owner is the most important receiver of feedback that reports issues regarding change history for individual requirements, meanwhile they are of relatively equal importance when it comes to sets of requirements. Similarly, feedback that reports issues regarding language are most relevant to the author of requirements at all timings, both for sets and individual requirements.

In most cases, the author of requirements should be of highest priority when it comes to receiving feedback regarding interdependencies and technical feedback, however requirement responsibles/owners follow closely behind.

These results show at which timing it can be applicable to deliver feedback to the relevant receivers. However, further investigation is required to gain knowledge if the specific feedback type should be delivered to the receiver at all the relevant timings, or if certain frequency of feedback or combination of those timings are optimal.

	ID 1: Relation to other parts of the system	Author of requirement	Test engineers	Requirements responsible / owner	Development team	Product owner	I don't know
Individual	During writing	90%	30%	70%	20%	30%	0%
	When reviewing	55%	20%	60%	15%	15%	10%
	Git commit	95%	15%	65%	5%	15%	0%
	Pushing	80%	10%	80%	15%	25%	0%
	Maintenance checks	40%	10%	60%	10%	20%	15%
	Any time	65%	30%	70%	30%	35%	15%
Sets	During writing	90%	20%	75%	15%	25%	0%
	When reviewing	55%	20%	70%	20%	20%	10%
	Git commit	100%	5%	75%	5%	15%	0%
	Pushing	85%	10%	85%	10%	35%	0%
	Any time	65%	35%	65%	35%	35%	15%

(a) Timings and receivers for User Story ID 1.

	ID 2: Broken links	Author of requirement	Test engineers	Requirements responsible / owner	Development team	Product owner	I don't know
Individual	During writing	95%	10%	45%	5%	0%	0%
	When reviewing	65%	0%	55%	0%	5%	10%
	Git commit	100%	0%	60%	10%	0%	0%
	Pushing	85%	0%	75%	0%	10%	0%
	Maintenance checks	50%	5%	65%	0%	10%	15%
	Any time	70%	10%	60%	10%	10%	15%
Sets	During writing	95%	5%	50%	0%	0%	0%
	When reviewing	65%	0%	60%	0%	5%	10%
	Git commit	100%	0%	60%	0%	10%	0%
	Pushing	85%	0%	70%	0%	10%	0%
	Any time	65%	10%	55%	10%	10%	15%

(b) Timings and receivers for User Story ID 2.

	ID 3: Requirement is a precursor to another requirement	Author of requirement	Test engineers	Requirements responsible / owner	Development team	Product owner	I don't know
Individual	During writing	95%	20%	55%	10%	5%	5%
	When reviewing	60%	5%	50%	5%	10%	15%
	Git commit	95%	0%	55%	0%	10%	5%
	Pushing	75%	5%	50%	10%	15%	5%
	Maintenance checks	45%	0%	50%	0%	10%	15%
	Any time	60%	5%	50%	10%	15%	15%
Sets	During writing	95%	10%	55%	10%	5%	5%
	When reviewing	55%	10%	50%	10%	10%	15%
	Git commit	95%	0%	60%	0%	10%	5%
	Pushing	75%	5%	55%	5%	15%	5%
	Any time	65%	10%	60%	15%	15%	15%

(c) Timings and receivers for User Story ID 3.

	ID 4: Placement of requirements	Author of requirement	Test engineers	Requirements responsible / owner	Development team	Product owner	I don't know
Individual	During writing	90%	0%	60%	5%	10%	5%
	When reviewing	65%	0%	60%	5%	15%	10%
	Git commit	95%	5%	70%	0%	20%	5%
	Pushing	85%	0%	75%	0%	10%	0%
	Maintenance checks	45%	0%	65%	0%	10%	15%
	Any time	70%	10%	60%	15%	15%	15%
Sets	During writing	90%	0%	60%	0%	10%	5%
	When reviewing	65%	0%	65%	0%	15%	10%
	Git commit	95%	0%	65%	0%	10%	5%
	Pushing	75%	0%	60%	0%	15%	5%
	Any time	70%	10%	60%	15%	15%	15%

(d) Timings and receivers for User Story ID 4.

	ID 5: Heat Map/Graph	Author of requirement	Test engineers	Requirements responsible / owner	Development team	Product owner	I don't know
Individual	During writing	90%	30%	75%	30%	40%	0%
	When reviewing	55%	25%	65%	30%	20%	10%
	Git commit	85%	10%	70%	10%	20%	0%
	Pushing	75%	15%	80%	30%	25%	0%
	Maintenance checks	40%	20%	65%	20%	25%	15%
	Any time	70%	40%	70%	40%	40%	15%
Sets	During writing	85%	20%	75%	20%	35%	0%
	When reviewing	55%	20%	65%	20%	20%	10%
	Git commit	85%	15%	80%	15%	25%	0%
	Pushing	75%	15%	85%	20%	30%	0%
	Any time	65%	40%	70%	40%	40%	15%

(e) Timings and receivers for User Story ID 5.

	ID 6: Forward reference	Author of requirement	Test engineers	Requirements responsible / owner	Development team	Product owner	I don't know
Individual	During writing	85%	0%	30%	0%	0%	15%
	When reviewing	60%	0%	35%	5%	5%	20%
	Git commit	90%	0%	40%	0%	5%	10%
	Pushing	70%	0%	50%	5%	10%	10%
	Maintenance checks	45%	0%	55%	0%	5%	15%
	Any time	65%	5%	45%	15%	10%	15%
Sets	During writing	85%	0%	40%	0%	10%	10%
	When reviewing	60%	5%	45%	5%	5%	15%
	Git commit	90%	0%	45%	0%	5%	10%
	Pushing	70%	0%	45%	0%	10%	10%
	Any time	65%	5%	55%	15%	10%	15%

(f) Timings and receivers for User Story ID 6.

Figure 5.5: Heat maps showing practitioners' ratings of specific receivers getting feedback at different timings for feedback types 1-6. Each row represents a timing (based on evaluation workshop in Cycle II) while each column represents a receiver of feedback. The values in the table represent the percentage of participants that approved of a receiver getting feedback at the corresponding timing. The darker the colour displayed in the table, the higher the correlation between a timing and a receiver for the relevant feedback type.

	ID 7: A requirement was changed	Author of requirement	Test engineers	Requirements responsible / owner	Development team	Product owner	I don't know
Individual	When reviewing	60%	35%	70%	35%	10%	10%
	Any time	75%	45%	80%	45%	40%	15%
Sets	Maintenance checks	40%	25%	75%	25%	30%	10%
	Any time	70%	40%	70%	40%	45%	15%

(a) Timings and receivers for User Story ID 7.

	ID 8: A list of the change history of a requirement	Author of requirement	Test engineers	Requirements responsible / owner	Development team	Product owner	I don't know
Individual	When reviewing	60%	5%	55%	0%	5%	15%
	Maintenance checks	40%	5%	65%	5%	5%	15%
	Any time	65%	25%	70%	25%	25%	15%
Sets	Maintenance checks	40%	5%	70%	5%	15%	10%
	Any time	70%	25%	70%	25%	20%	15%

(b) Timings and receivers for User Story ID 8.

	ID 9: How often a requirement has changed	Author of requirement	Test engineers	Requirements responsible / owner	Development team	Product owner	I don't know
Individual	When reviewing	55%	0%	65%	0%	10%	10%
	Maintenance checks	45%	25%	70%	25%	30%	10%
	Any time	70%	25%	75%	25%	20%	15%
Sets	Maintenance checks	40%	0%	70%	0%	10%	10%
	Any time	70%	25%	70%	25%	25%	15%

(c) Timings and receivers for User Story ID 9.

	ID 10: A requirement was removed	Author of requirement	Test engineers	Requirements responsible / owner	Development team	Product owner	I don't know
Individual	When reviewing	50%	35%	65%	35%	20%	10%
	Maintenance checks	45%	30%	70%	30%	25%	10%
	Any time	75%	45%	80%	45%	40%	15%
Sets	Maintenance checks	40%	25%	75%	25%	30%	10%
	Any time	70%	40%	70%	40%	45%	15%

(d) Timings and receivers for User Story ID 10.

	ID 11: This requirement has changed recently	Author of requirement	Test engineers	Requirements responsible / owner	Development team	Product owner	I don't know
Individual	Maintenance checks	45%	25%	70%	25%	30%	10%

(e) Timings and receivers for User Story ID 11.

	ID 13: Consistent usage of language	Author of requirement	Test engineers	Requirements responsible / owner	Development team	Product owner	I don't know
Individual	During writing	95%	5%	55%	0%	0%	0%
	When reviewing	65%	5%	60%	5%	5%	10%
	Git commit	100%	0%	50%	0%	5%	0%
	Any time	70%	10%	50%	10%	10%	15%
Sets	Any time	65%	10%	45%	10%	10%	15%

(f) Timings and receivers for User Story ID 13.

Figure 5.6: Heat maps showing practitioners' ratings of specific receivers getting feedback at different timings for feedback types 7-13. Each row represents a timing (based on evaluation workshop in Cycle II) while each column represents a receiver of feedback. The values in the table represent the percentage of participants that approved of a receiver getting feedback at the corresponding timing. The darker the color displayed in the table, the higher the correlation between a timing and a receiver for the relevant feedback type.

	ID 14: Quality of language	Author of requirement	Test engineers	Requirements responsible / owner	Development team	Product owner	I don't know
Individual	During writing	95%	5%	55%	0%	5%	0%
	When reviewing	55%	0%	60%	0%	15%	10%
	Git commit	100%	0%	45%	0%	5%	0%
	Pushing	75%	0%	50%	0%	10%	5%
	Maintenance checks	45%	0%	65%	0%	15%	15%
	Any time	70%	10%	50%	10%	10%	15%
Sets	Any time	70%	10%	55%	10%	10%	15%

(a) Timings and receivers for User Story ID 14.

	ID 15: Templates	Author of requirement	Test engineers	Requirements responsible / owner	Development team	Product owner	I don't know
Individual	During writing	85%	0%	50%	0%	0%	0%
	When reviewing	65%	0%	60%	0%	0%	10%
	Git commit	100%	0%	55%	0%	5%	0%
	Pushing	75%	0%	50%	0%	10%	5%
	Any time	75%	5%	50%	10%	10%	15%
Sets	Any time	65%	5%	50%	10%	10%	15%

(b) Timings and receivers for User Story ID 15.

	ID 16: Requirement is too long	Author of requirement	Test engineers	Requirements responsible / owner	Development team	Product owner	I don't know
Individual	During writing	90%	5%	40%	10%	5%	5%
	When reviewing	60%	0%	50%	5%	5%	10%
	Git commit	100%	0%	45%	0%	5%	0%
	Pushing	75%	0%	45%	0%	10%	5%
	Any time	70%	15%	50%	15%	15%	15%
Sets	Any time	70%	15%	50%	15%	15%	15%

(c) Timings and receivers for User Story ID 16.

	ID 17: Passive language	Author of requirement	Test engineers	Requirements responsible / owner	Development team	Product owner	I don't know
Individual	During writing	85%	5%	25%	0%	0%	10%
	Git commit	95%	0%	35%	0%	5%	5%
	Pushing	75%	0%	50%	0%	10%	5%
	Any time	70%	10%	50%	10%	10%	15%
Sets	Any time	70%	10%	50%	10%	10%	15%

(d) Timings and receivers for User Story ID 17.

	ID 18: Validation	Author of requirement	Test engineers	Requirements responsible / owner	Development team	Product owner	I don't know
Individual	When reviewing	85%	0%	50%	0%	0%	0%
	Git commit	95%	15%	50%	10%	15%	5%
	Pushing	80%	15%	60%	15%	20%	5%
	Maintenance checks	40%	15%	50%	10%	15%	15%
	Any time	75%	35%	55%	30%	30%	15%
Sets	Pushing	80%	20%	55%	20%	20%	5%
	Any time	70%	40%	60%	35%	35%	15%

(e) Timings and receivers for User Story ID 18.

	ID 19: Testability of requirements	Author of requirement	Test engineers	Requirements responsible / owner	Development team	Product owner	I don't know
Individual	When reviewing	60%	40%	40%	15%	15%	10%
	Git commit	95%	35%	75%	10%	20%	0%
	Pushing	75%	45%	75%	25%	20%	0%
	Maintenance checks	45%	40%	60%	10%	15%	15%
	Any time	75%	50%	60%	40%	35%	15%
Sets	Pushing	75%	45%	70%	25%	20%	0%
	Any time	70%	50%	60%	40%	35%	15%

(f) Timings and receivers for User Story ID 19.

	ID 20: Quality scoring system	Author of requirement	Test engineers	Requirements responsible / owner	Development team	Product owner	I don't know
Individual	When reviewing	55%	0%	60%	0%	15%	10%
	Git commit	90%	5%	60%	0%	10%	5%
	Pushing	80%	5%	70%	0%	10%	5%
	Maintenance checks	45%	0%	65%	0%	15%	15%
	Any time	75%	25%	55%	25%	30%	15%
Sets	Any time	65%	25%	55%	25%	25%	15%

(g) Timings and receivers for User Story ID 20.

	ID 21: Duplicated requirements in any specification	Author of requirement	Test engineers	Requirements responsible / owner	Development team	Product owner	I don't know
Individual	When reviewing	65%	5%	60%	10%	10%	10%
	Git commit	100%	0%	70%	0%	15%	0%
	Pushing	80%	0%	65%	5%	10%	0%
	Maintenance checks	45%	0%	65%	0%	15%	15%
	Any time	75%	25%	70%	30%	25%	15%
Sets	During writing	95%	10%	70%	5%	5%	0%
	When reviewing	65%	10%	60%	5%	10%	10%
	Git commit	100%	0%	70%	0%	10%	0%
	Pushing	80%	0%	65%	0%	15%	0%
	Maintenance checks	55%	5%	70%	5%	15%	10%
	Any time	70%	25%	70%	30%	25%	15%

(h) Timings and receivers for User Story ID 21.

Figure 5.7: Heat maps showing practitioners' ratings of specific receivers getting feedback at different timings for feedback types 14-21. Each row represents a timing (based on evaluation workshop in Cycle II) while each column represents a receiver of feedback. The values in the table represent the percentage of participants that approved of a receiver getting feedback at the corresponding timing. The darker the colour displayed in the table, the higher the correlation between a timing and a receiver for the relevant feedback type.

5.2.6 Abstraction Levels

The abstraction levels of automated feedback discovered to be most relevant when making parallel changes are listed below.

Error
Feedback should be given at the abstraction level of an error when something is completely wrong.

> *"Hard error messages should only be given out if there is really something terribly wrong, otherwise it can be mostly information or warning."*
>
> *-Interviewee 3*

This suggests that errors should only be given when the changes should not be pushed or accepted into the workflow at all. Errors should indicate a full stop and ban further proceeding without the reported issue being fixed.

Warning
Feedback is on the abstraction levels of warnings when there is something that could be wrong in some way, but is tolerated to be pushed or accepted into the workflow, as mentioned by interviewee three.

> *"A warning would be for me something that is kind of wrong, but is kind of tolerated."*
>
> *-Interviewee 3*

Fixing those issues should be taken into consideration, but interviewees mentioned that there might be a valid reason for ignoring the warning.

Information
Information is an abstraction level of feedback that should give instructions or advice on how to make the quality of requirements higher.

> *"Information is a suggestion that doesn't necessarily have to be implemented."*
>
> *-Interviewee 4*

Interviewees further mentioned that information can be useful or important to know about when working with the corresponding requirement or requirements specification and might save the reviewer's time later in the process. However, as mentioned by interviewee four, information does not necessarily have to be implemented.

Scoring System
Scoring system is not one specific abstraction level but when a score is given on the quality of requirements. This can for example be a score on language quality, similarity of requirements, or a score on the criticality or severity of the requirements. A high score indicates that the requirement has high quality, contrary to a lower score that indicates that the requirement is of low quality.

> *"I think it is sufficient to get the a score on it. And, then I get a score I say, OK, this does not look so good. Maybe I should change it. I trust that [we] notice that 'OK, we got the bad score here' and then we can make a good judgement of whether this was actually correct [feedback]."*
>
> *-Interviewee 5*

During discussions of scoring systems it was mentioned as a solution allowing and trusting the requirements engineers to decide on the quality of the requirements themselves while having the automatic scoring for reference, since an automatic system might not work perfectly in every situation.

5.3 RQ3 (Evaluation)

How valuable are the proposed solutions as rated by practitioners in regards to supporting common problems caused by a lack of automated requirements feedback in agile systems?

The goal of the study was to discover types of feedback that can support common problems within requirements engineering that can be avoided by giving automated feedback. To investigate if the solutions are truly helpful, they were evaluated with requirements engineers and specialists from the software industry. This was done to know how valuable the proposed solutions are to those who matter the most. The outcome is a prioritized list of the solutions that indicates their business value as well as implementation cost.

5.3.1 Feedback Types

Table 5.1 shows the business value and implementation cost of each of the proposed solutions, the feedback types. Those feedback types are all proposed solutions to support common problems caused by a lack of automated requirements feedback in agile systems. The values in the table are ordered in descending business value order, where the highest number indicates the highest value. Solutions with the same business value are again internally ordered in a descending order based implementation cost of implementing the feedback type, where the highest number indicates the highest cost of implementation. The business value ranges from 1 - 3 (small, medium, high) while the cost of implementation ranges from 1 - 4 (small, medium, large, huge). Solutions with the exact same values for both cost and value are rated equally.

The findings show that there are five solutions that were considered the most valuable by practitioners in regards to supporting the common problems discussed in RQ1. The most valuable solutions were: broken links, list of the change history of a requirement, automatic validation possibilities, relation to other parts of the system and automatically detecting the testability of requirements.

When asked to order the four automatic feedback types, in the order of value, the practitioners ranked them in the following order: feedback for interdependencies, feedback for change history, language feedback and technical feedback.

These results do not correspond fully to the rating of the feedback types done by the same practitioners, in the same workshop. There, two of the top five are feedback for interdependencies while the other three are technical feedback. This might be an indication to further research the categorization of the feedback types.

The rated cost of implementation provides an indication of the feasibility of implementing the solutions although they should be taken with a grain of salt, since the focus was on involving requirements engineering experts evaluating the solutions rather than people with enough knowledge of how existing tools are implemented. Therefore, their competence when it comes to rating the cost of implementation might vary.

Table 5.1: Business value and implementation cost of each of the proposed solutions, the feedback types, based on rating by practitioners during final evaluation workshop.

ID	Feedback Types	Value (1-3)	Cost (1-4)
2	Broken links	3	1
8	A list of change history of a requirement.	3	2
18	Validation	3	2
1	Relation to other parts of the system	3	3.5
19	Testability of requirements	3	4
15	Templates	2	1
9	How often a requirement has changed	2	1
3	Requirement is a precursor to another requirement	2	1
7	This requirement has changed recently.	2	1
12	A requirement was created	2	1
13	Consistent usage of language	2	2
14	Quality of language	2	2
6	Forward Reference	2	2
4	Placement of requirements	2	2
5	Heat Map/Graph	2	3.5
16	Requirement is too long	1	1
17	Passive Language	1	1
10	A requirement was removed	1	2
11	The requirement has changed recently.	1	2
21	Duplicated requirements in any specification.	1	2
20	Quality Scoring System	1	3

Figure 5.8 provides visual representation of the business value of each of the feedback types, plotted against the cost of implementation. For the business value, the higher the data point is on the axis, the higher the value of the proposed solution, as rated by practitioners. When looking at the cost of implementing, the data point being further to the right on the graph indicates a higher implementation cost. Looking at both value and cost, the top left quadrant is the most valuable option having high business value and a low cost of implementation. Most of the solutions were positioned in or at the edge of the aforementioned quadrant, deeming them valuable in regards to supporting common problems caused by a lack of automated requirements feedback in agile system. Data points in clusters are rated equally.

When asked if user stories are useful guidelines to solve the problems, the practitioners were positive that they are. Further it was mentioned that the mapping between the solutions and the problems are especially useful.

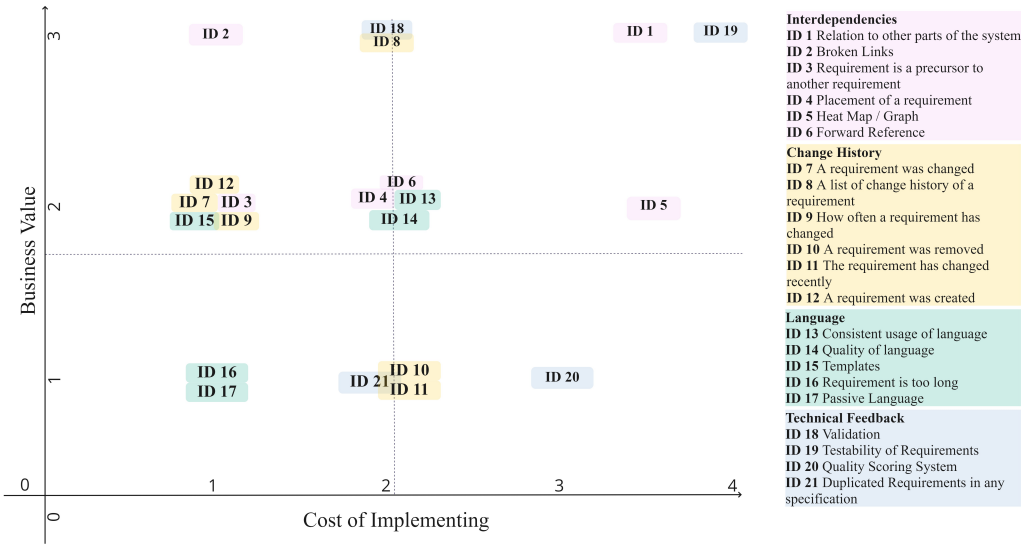


Figure 5.8: Value-Cost matrix displaying all feedback types based on rating during a workshop with practitioners. The colors of the user stories indicate the feedback category they belong to (pink = Interdependencies, yellow = Change History, green = Language, and blue = Technical Feedback).

6
Discussion

This chapter discusses the findings of the research, interprets them and explains their implications for both practitioners and researchers. Finally, the chapter discusses the limitations of the research.

6.1 Interpretations of the Findings

The research identified multiple problems of requirements verification and validation that can be addressed by using automated feedback. The problems are split into the categories of problems related to interdependencies, change history, and technical problems, e.g., validation and testability of requirements (RQ1). Potential solutions to those problems were collected and put in the same categories. However, solutions might address problems that are part of a different category. The solutions are presented as guidelines that support practitioners and researchers when improving the quality of requirements in their systems (RQ2). All of the proposed solutions were rated valuable by practitioners to support the problems identified (RQ3).

The results indicate that the proposed solutions (RQ2) are valuable (RQ3) when it comes to solving the identified problems of requirements verification and validation (RQ1) that can be solved with automation.

The receivers of feedback as well as the timings are of high importance when it comes to the feedback itself being of value, which is reflected in the findings of the study which indicate how valuable each solution is in regards to the problems it solves. Subsequently, the abstraction levels of the feedback should be considered in relation to the receivers and timings.

The guidelines support continuous requirements validation and thus increase their quality from early on in the process. The continuous validation reduces the likelihood of issues which reduces the cost of reworking requirements. The findings of the research can, when applied, lead to improved quality of requirements and specifications resulting in increased software quality. The guidelines of how to apply the feedback provide a tangible version of the research findings and demonstrate them to practitioners and researchers.

Users of the guidelines must consider their own contexts and make critical decisions about where to apply the respective solutions based on their specific problems, goals and competence. The guidelines should be used as a reference when applying automated feedback to workflows in large-scale agile environments. This is supported by the fact that the research findings show that it can be unclear when to give each type of feedback and to whom, but there are options to consider that are

supplied as valuable options as starting points when applying automated feedback to requirements management tools.

The findings of the study support Post and Fuhr's claim [14] that semantic checks for consistency and completeness are important factors of requirements quality and that syntactical rules alone are not sufficient for evaluating requirements quality. However, this thesis does not differentiate as distinctly between semantic and syntactical rules as Post and Fuhr do, as the results of this thesis recognize syntactical features, such as templates and language use, to be contributors to the overall quality and consistency of requirements. Despite that, Post and Fuhr's main notion of semantic checks being more valuable is supported in the findings of this thesis when it is considered that experts ranked broken links, change history, validation, and testability higher than any language-related feedback.

This thesis adds to the existing work done for the textual requirements management tool T-Reqs [6], [13], where the contribution of this thesis is that it suggests which automatic feedback types to incorporate into the existing tool. These suggestions are made through the guidelines that link feedback types to their timings and receivers. This thesis supports the findings of Kasauli et al. [5] that there is a ground for incorporating validation of requirements into the workflow of organizations' version control systems.

Comparing this thesis to Singh's [26] reveals that the two studies tackle similar problems and ideas. While this thesis focused on identifying challenges within requirements verification and validation as they are perceived by requirements specialists, Singh's study suggested machine learning and natural language processing as ways of identifying themes in how requirements have been written as well as validating flaws in requirements after they have been reported. The main difference between this thesis and Singh's is that this thesis presents the value of implementing certain feedback types into requirements management tools, as well as when feedback should be displayed and to whom, while Singh focuses on suggesting how certain automated feedback can be implemented using machine learning and natural language processing.

Moreover, a comparison of this thesis's findings reveals there are differences between the T-Reqs tool, when it has been extended with the suggested solutions of the thesis, and the tools commonly used in industry as T-Reqs has clear advantages over the common tools. First and foremost, extending T-Reqs with the suggested solutions would allow it to provide a wide range of support when writing and managing requirements in an agile way as its git-based structure allows for easy collaboration between individuals and teams. It further supports the integration of feedback into the continuous integration process and the development life cycle, as the feedback is already integrated into and suitable to git.

By contrast, most of the other existing tools, perhaps with the exception of Jira, are not so well suited to agile ways of working, especially in large-scale agile development, since the teams do not always have direct access to the tools and therefore have to export the requirements out of the tools in order to work with the requirements. That procedure does not fit agile ways of working very well since that introduces inconsistencies between requirements when they are taken out of the system and potentially altered in some way, and the teams don't know what other

teams have done with the requirements.

T-Reqs has the potential to combine the most useful practices of the other tools, while also having the potential of providing automatic feedback for requirement changes, including how interdependencies are managed, evaluating the quality of requirements and supporting requirements validation in the context of continuous integration life cycles.

6.1.1 Receivers

The findings show that the most important receivers are the author of the requirement and the requirements responsible/owner. The chosen threshold was that more than 50% of survey respondents selected the option for the receiver to get certain feedback at a certain time, meaning that more than half of respondents had to have selected the option for it to be considered significant. For the other receivers, none of them scored higher than 45% for any feedback type at any timing.

What this indicates is that authors and responsibles of requirements are in general those who are closest to the requirements work and as such should be considered of priority when choosing receivers of feedback.

6.1.2 Timings

The study analysed the timings of which feedback should be given as well as which timings are suitable for each feedback type in relation to who should receive the feedback at that timing.

According to the findings, every timing is considered relevant for more than one feedback type. When considering which feedback types a timing is appropriate for, the timing *Any time* appears more frequently than any of the other timings, being considered relevant for all but one feedback type. However, *Any time* was in general not considered a very automatic event, since it related to a user's ability to access feedback at any time, e.g. when researching requirements and wanting to get insights into the interdependencies, change history, language use or other technical feedback related to the requirement or requirements. The second most common timing for individual requirements was *When reviewing*, which appeared for a total of 17 different feedback types. For sets of requirements, *Pushing* had the second most feedback types associated with it, appearing 10 times.

The timing *Maintenance checks*, which refers to system-wide tests that can be triggered manually or automatically at certain intervals, was the main deviation from the receivers of feedback primarily being the author of a requirement, and the requirements responsible/owner being the second most important receiver of feedback. For *Maintenance checks* of both individual requirements and sets of requirements, the roles were reversed as requirements responsible was considered of higher priority than the author of the requirement.

The findings of the study present at which timing it can be applicable to deliver feedback to relevant receivers. However, the question remains of which combination of the timings is desirable, i.e. if the feedback should be delivered at all the corresponding timings or if there is a specific combination or frequency of feedback that

is optimal to reach higher quality in requirements.

6.1.3 Abstraction Levels

The abstraction levels of information, warning, and error were suggested based on literature review and interview responses. However, the participants of the study did not agree which abstraction levels were appropriate for the different kinds of feedback, and they had differing opinions on which abstraction levels are appropriate for the different timings of the feedback types. The findings also indicate the importance of knowing the receiver of particular feedback when deciding the abstraction level.

For example, some of the interviewed requirement experts wanted to get errors on certain things, especially when it involved an area they had a particular interest in and wanted their colleagues' work to be of the same standard as theirs. During the interviews and workshops, other experts highlighted that they did not want anything to be a hard error since they claimed to have more trust in their co-workers than in an automatic tool since there could always be unforeseen exceptions to rules set by the requirements tool. Discussions also emerged regarding it generally being most valuable to get feedback as soon as possible so that faults are fixed early. Additionally, it might not be helpful to be warned too frequently about the same things. When the feedback is ignored for some reason, it might create noise in the feedback when the requirements engineer is informed about the same potential faults repeatedly.

One way to mitigate the problem of having different abstraction levels that causes a varying degree of disruption to one's workflow, is providing a requirements quality score instead of an error or warning. The higher the score, the better the quality of the requirement according to the standards implemented in that specific tool's ruleset. Thus, a quality scoring system for requirements could be used to inform the users of the automated requirements tool about the things that can be improved by showing them a score that indicates whether the requirement looks good or poor, and even that the scores for the different elements of the requirement are visible so that the user of the tool knows what can be improved in the requirement.

6.2 Implications to Practice

This research provides practitioners with guidelines of which of their day-to-day problems are solvable by automation of processes, as the study analyzed problems that currently exist in industry (RQ1). Further, it provides them with guidelines of possible solutions (RQ2) to those problems, evaluated by expert individuals in terms of how valuable they are to industrial work environments (RQ3).

Following the guidelines of which feedback to automate, where to place it in their workflows and whom to send the feedback supports practitioners in applying those guidelines to their current tools or creating new powerful automated tools to better their requirements processes. Using those guidelines will help them prioritize, based on the problems they would like to solve, which feedback types it is feasible for them to solve. Further, practitioners have the possibility of prioritizing their selection of

feedback by taken into consideration the rating of business value (RQ3), done by experts in the field. Using such tools can provide them in discovering problems in requirements early, and thereby reducing the manufacturing of malfunctioning products that are never used, aiding companies to become more sustainable.

The research further supports sustainable working environment by aiding remote work. This decreases the necessity of having to travel between places, or even countries, for short periods with a huge cost on the environment.

6.3 Implications to Research

This study provides value to research by identifying problems (RQ1) that can be solved by automating requirements feedback (RQ2), and connecting them to who should receive feedback and when the feedback should be given. An investigation has been done on abstraction levels with the results that they are something that need to be investigated further. The study provides grounds for further research in multiple directions, such as in terms of abstraction levels and how they are not clear cut, as well as looking further into the frequency of feedback at each point in time when giving the feedback in the work process.

Since the solutions have been ranked by practitioners (RQ3) and are demonstrated in guidelines, it is easy for researchers to apply those on tools and processes before conducting further research.

The study has identified the need for further investigation of multiple different aspects of the research such as abstraction levels and frequency of feedback at each point in time in the workflow. Further, the research has put priorities on the business value of the proposed solutions while providing insights on the feasibility of implementing them. This provides grounds for investigating further, which of the proposed solutions are a feasible option in near future and which of them become too expensive in relation to their business value when applied in practice.

Furthermore, the research artifact, the guidelines, provide valuable suggestions on how to further develop the open-source variant of T-Reqs. The guidelines have been evaluated (RQ3) and therefore give valuable suggestions of the prioritization of the suggested feedback types and when to give them, i.e. where to locate them in the git-based workflow.

6.4 Limitations

This section discusses the threats to validity of the research based on the classification of construct validity, internal validity, external validity and reliability proposed by Runeson and Höst [41].

6.4.1 Construct Validity

A threat to the construct validity of the research is the aspect of design science research which is based on showing the solutions to the participants and then asking questions about it. However, while this is something that can be leading, the research

benefits highly from other aspects of the methodology, e.g. getting feedback, ideas, and concerns from experts in an iterative manner, which is one of the benefits of conducting research following Design Science Research.

Another threat to the construct validity of the research is that strong opinions of individuals that are a part of the validation and evaluation process could lead to aspects being removed as feasible parts of the solution. This was mitigated by being critical but at the same time open-minded when analyzing the collected data for solution candidates and implementations.

Since the artifact only really materialized at the end of the research, it is debatable whether design science research was the best research method to base this study on. The method made the purpose of the different steps of the cycles a bit ambiguous, especially when it came to solution validation when the solutions were still in the form of written down ideas. Despite that, it was valuable to iterate the findings. Conducting the study as design science research, provided the opportunity to constantly validate the findings and add new problems and solutions as they arose. That balances the threat to the construct validity, which is valuable when evaluating if findings are representative of different individuals' opinions or only one individual's perspective. Therefore, conducting design science research was arguably more valuable than for example a case study in this context, due to the fact that the study relied on discovering issues, solving them and confirming that the solutions are good. Thus, iterating the solutions was a way of delivering higher quality solutions in the same amount of time as a case study would take.

Another threat to the construct validity is that there are terms used in the research that can have multiple different definitions in the context of software engineering, such as completeness, consistency and validation. This can lead to misunderstandings between the researchers and participants of the study and was therefore mitigated by having definitions on hand when necessary and explaining them carefully when appropriate.

6.4.2 Internal Validity

The main threat to internal validity was that the used sampling method for the interviews and workshops was convenience sampling, i.e. participants who were accessible to the researchers were recruited for the interviews [32], and the sampling was mainly done at one company. Consequently, the research was in the context of a single requirements management tool, T-Reqs. Collecting data mainly with a single company makes it so the focus can be mainly on their processes, roles, etc. However, most of those company-specific aspects are general and used elsewhere in the software engineering industry, so it is not a significant threat to the research.

This was further mitigated by further bringing in opinions of developers of the open-source variant of T-Reqs to get a broader range of opinions and incorporating other participants through the Software Center Project. This was done especially during workshops and the evaluation survey at the end of the research process. The sampling method of snowball sampling, where an initial group of requirements specialists were sent the survey and asked to forward it to anyone who might be interested [32], involved the threat of not knowing the sample size or demographics

of those who received the survey. This was mitigated by including questions in the survey that helped identify the characteristics of those who filled in the survey, and therefore represent the individuals whose experience and opinions the data is based.

The diversity of years of experience of the interviewees can be seen as a threat to internal validity. However, they are all typical users of the requirements maintenance tools and are highly incorporated in the process of creating, changing, and maintaining requirements. Thereby, this was not seen as a significant threat to the internal validity of the research.

Additionally, there was awareness that conducting a survey can introduce risk in terms of the number of answers collected. This was mitigated by including individuals that have already been interviewed or showed interest in the thesis and were likely to give their answers and spread the survey to their colleagues. Another trade-off of sending out a survey is the difficulty of balancing the combination of respondents and their identified roles. Unsurprisingly, the top-ranked receivers of feedback correlated with the most common roles of the respondents. Therefore, it is an important mitigation strategy to take these aspects into account when discussing the research results. However, the two highest-ranking receivers were in line with previous observations made through the interviews about which of the receivers have the most responsibility when it comes to creating and maintaining requirements. A survey was the chosen data collection method for receivers since it was considered the most feasible option to get quantitative data. It was used for solution validation at the beginning of Cycle III. It was considered the option that would give the most valuable results in terms of the information needed at that point in the process.

6.4.3 External Validity

Even though the research is based on T-Reqs, the suggested guidelines are generalizable because they follow git-based processes, which are common within the software engineering industry. The feedback types, timings in the workflow, and the receivers of feedback are all defined in a way that can be mapped to other tools or processes in software engineering environments, which minimizes the threats to external validity. Practitioners at other companies can use the guidelines as suggestions to incorporate the findings into their own systems.

Further, incorporating individuals from different companies than the main telecommunications company for the research ensures that the data is generalizable and not only relevant to their isolated context. The findings have been represented and evaluated by individuals from different backgrounds and companies, which further supports this argument.

6.4.4 Reliability

A threat to the reliability of the research is that both interviews and workshops were semi-structured, which makes them harder to replicate. This, however, provided great value to the research results as it led to insightful discussions and provided valuable data collection. Having a clear description of the method used in all cycles and providing both interview guides (Appendix A and Appendix B) as well as the

survey (Appendix C) mitigates this risk and supports other researchers in collecting the same data.

To support the reliability of the research, emphasis was put on the diversity of individuals evaluating the findings of each cycle, i.e. no individual person evaluated their own proposed solutions alone. All six interviewees were invited to the final workshop, and three of them participated in the workshop. This was not considered a risk since this gave them a chance to reflect upon if the proposed solutions were relevant to the problems they had experienced. This was done to make sure that the results are based on multiple individuals' experts' opinions and can be reproduced with individuals with similar roles.

6.4.5 Ethics

There were no specific ethical issues raised by interviewees or workshop participants during the research. However, ethical aspects to pay attention to are that it is essential to be careful and critical when automating processes, especially those most commonly done manually by humans. The reason being is that people might trust the processes blindly, which, if the processes are not automated correctly and carefully, can inevitably lead to mishaps. Depending on the industry using automated tools for requirements, it could have serious implications if the tool in any way causes an incorrect product or a faulty product being produced. Additionally, an ethical issue worth mentioning is the effect it can have on individual jobs if processes are automated that have previously been most commonly done manually.

7
Conclusion

In agile systems development, there is a need for automatically providing feedback for requirement changes. As organizations grow in an ever-changing environment, they must respond to change flexibly and plan continuously, which makes requirements management more complex.

By investigating literature and consulting with experts in the field of requirements engineering, challenges related to the verification and validation of requirements were identified, and solutions to them suggested. The solutions are presented as user stories that connect feedback types to requirements verification and validation challenges that they aim to solve. The solutions are rated by practitioners based on business value and implementation cost, consequently providing a value to each of the solutions. Suggestions of timings of where to place feedback in git-based workflows and information of the receivers to whom the feedback should be presented give guidance on how to apply the feedback for it to be valuable. Discussions of abstraction levels in the context of giving automated feedback can aid the user in deciding which abstraction level to consider when applying the feedback.

Providing validated guidelines, rated by experts, on how to apply the automatic requirements feedback to requirements management tools, such as T-Reqs, provides a field for large-scale agile companies to address problems that arise when changes are made to requirements in parallel. Exploring the problems and solutions iteratively through the steps of the regulative cycle has provided extensive guidelines for both practitioners and researchers on how to apply the findings in their contexts.

The fact that experts have validated the guidelines in terms of the business value they provide and an estimate of implementation cost gives assurance to those applying the guidelines. The validation of the guidelines ensures that the proposed solutions are a valuable aid for solving the problems they map to and giving an idea of the feasibility of implementing those solutions.

The study findings have provided information on where further research is needed. Based on these, recommendations to future researchers are to consider the guidelines and implement and apply the solutions to requirements managing tools, such as T-Reqs. When applying the solutions, one should quantitatively examine and validate the findings of this qualitative research by analyzing how well the proposed solutions work in practice.

It would be beneficial to evaluate the findings by implementing them into the tool, and then conducting a comparison study to see whether and how much requirements quality improves in a company, by comparing the quality of requirements before and after adapting the updated tool.

There is a need to examine the frequency of feedback when they are delivered to

determine how often a specific receiver should be notified of the same issue, especially after having dismissed it before. The research also discovered the need to connect and investigate the effects of abstraction levels at each point in time when delivering feedback to specific receivers to understand if the same abstraction levels apply to all receivers at all times or if they should be different.

Bibliography

[1] F. Almeida and E. Espinheira, "Large-scale agile frameworks: A comparative review," *Journal of Applied Sciences, Management and Engineering Technology*, vol. 2, pp. 16–29, 2021. [Online]. Available: doi:10.31284/j.jasmet.2021.v2i1.1832.

[2] L. Gren and P. Lenberg, "Agility is responsiveness to change: An essential definition," 2020, pp. 348–353. [Online]. Available: doi:10.1145/3383219.3383265.

[3] S. Alsaqqa, S. Sawalha, and H. Abdel-Nabi, "Agile Software Development: Methodologies and Trends," *International Journal of Interactive Mobile Technologies (iJIM)*, vol. 14, pp. 246–270, 2020. [Online]. Available: doi:10.3991/ijim.v14i11.13269.

[4] I. Inayat, S. S. Salim, S. Marczak, M. Daneva, and S. Band, "A systematic literature review on agile requirements engineering practices and challenges," *Computers in Human Behavior*, 2014. [Online]. Available: doi:10.1016/j.chb.2014.10.046.

[5] 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*, vol. 172, p. 110 851, 2021. [Online]. Available: doi:10.1016/j.jss.2020.110851.

[6] E. Knauss, G. Liebel, J. Horkoff, *et al.*, "T-Reqs: Tool Support for Managing Requirements in Large-Scale Agile System Development," *2018 IEEE 26th International Requirements Engineering Conference (RE)*, pp. 502–503, 2018. [Online]. Available: doi:10.1109/RE.2018.00073.

[7] E.-M. Schön, D. Winter, M. Escalona, and J. Thomaschewski, "Key challenges in agile requirements engineering," 2017, pp. 37–51. DOI: 10.1007/978-3-319-57633-6_3.

[8] S. Jayatilleke and R. Lai, "A systematic review of requirements change management," *Information and Software Technology*, vol. 93, 2017. DOI: 10.1016/j.infsof.2017.09.004.

[9] N. Sekitoleko, F. Evbota, E. Knauss, A. Sandberg, M. Chaudron, and H. Olsson, "Technical dependency challenges in large-scale agile software development," *Lecture Notes in Business Information Processing*, vol. 179, 2014. DOI: 10.1007/978-3-319-06862-6_4.

[10] E. Collins, A. Dias-Neto, and V. F. d. Lucena Jr., "Strategies for agile software testing automation: An industrial experience," pp. 440–445, 2012. [Online]. Available: doi:10.1109/COMPSACW.2012.84.

[11] D. Ståhl and J. Bosch, "Cinders: The continuous integration and delivery architecture framework.," *Information and Software Technology*, vol. 83, pp. 76–93, 2018. DOI: 10.1145/3202710.3203165.

[12] D. Georgakopoulos, M. Hornick, and A. Sheth, "An overview of workflow management: From process modeling to workflow automation infrastructure," *Distributed and Parallel Databases*, vol. 3, pp. 119–153, 1995. DOI: 10.1007/BF01277643.

[13] treqs ng, "TReqs: Tool Support for Managing Requirements in Large-Scale Agile System Development," n.d. Accessed on: Jan. 6, 2022. [Online]. Available: https://gitlab.com/treqs-on-git/treqs-ng.

[14] A. Post and T. Fuhr, "Case study: How Well Can IBM's "Requirements Quality Assistant" Review Automotive Requirements?" In *REFSQ 2021 Workshops*, ser. CEUR Workshop Proceedings, vol. 2857, CEUR-WS.org, 2021. [Online]. Available: http://ceur-ws.org/Vol-2857/nlp4re8.pdf.

[15] C. Ebert and M. Paasivaara, "Scaling agile," *IEEE Software*, vol. 34, no. 6, pp. 98–103, 2017. [Online]. Available: doi:10.1109/MS.2017.4121226.

[16] M. Shahin, M. Ali Babar, and L. Zhu, "Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices," *IEEE Access*, 2017. [Online]. Available: doi:10.1109/ACCESS.2017.2685629.

[17] B. Fitzgerald and K. Stol, "Continuous software engineering: A roadmap and agenda," *Journal of Systems and Software*, vol. 123, pp. 176–189, 2017. [Online]. Available: doi:10.1016/j.jss.2015.06.063.

[18] R. Torkar, T. Gorschek, R. Feldt, M. Svahnberg, U. A. Raja, and K. Kamran, "Requirements traceability: A systematic review and industry case study," *International Journal of Software Engineering and Knowledge Engineering*, 2012.

[19] O. Ormandjieva, I. Hussain, and L. Kosseim, "Toward a text classification system for the quality assessment of software requirements written in natural language," in *Fourth International Workshop on Software Quality Assurance: In Conjunction with the 6th ESEC/FSE Joint Meeting*, New York, NY, USA: Association for Computing Machinery, 2007, pp. 39–45. DOI: 10.1145/1295074.1295082.

[20] E. Knauss, O. Brill, I. Kitzmann, and T. Flohr, "SmartWiki: Support for high-quality requirements engineering in a collaborative setting," 2009, pp. 25–35. [Online]. Available: doi:10.1109/WIKIS4SE.2009.5069994.

[21] L. Zhao, W. Alhoshan, A. Ferrari, *et al.*, "Natural language processing for requirements engineering: A systematic mapping study," *ACM Comput. Surv.*, vol. 54, no. 3, 2021, ISSN: 0360-0300. DOI: 10.1145/3444689.

[22] F. Dalpiaz, A. Ferrari, X. Franch, and C. Palomares, "Natural language processing for requirements engineering: The best is yet to come," *IEEE Software*, vol. 35, no. 5, pp. 115–119, 2018. DOI: 10.1109/MS.2018.3571242.

[23] E. D. Liddy, "Natural language processing," *Encyclopedia of Library and Information Science*, 2001.

[24] M. Fowler, *Refactoring: Improving the Design of Existing Code*, 2nd ed., ser. Addison-Wesley Signature Series (Fowler). Boston, MA, USA: Pearson Education, 2018.

[25] "Detecting terminological ambiguity in user stories: Tool and experimentation," *Information and Software Technology*, vol. 110, pp. 3–16, 2019. DOI: https://doi.org/10.1016/j.infsof.2018.12.007.

[26] M. Singh, "Automated validation of requirement reviews: A machine learning approach," in *2018 IEEE 26th International Requirements Engineering Conference (RE)*, 2018, pp. 460–465. DOI: 10.1109/RE.2018.00062.

[27] R. Wohlrab, P. Pelliccione, A. Shahrokni, and E. Knauss, "Why and how your traceability should evolve: Insights from an automotive supplier," *CoRR*, vol. abs/2005.09414, 2020. [Online]. Available: https://arxiv.org/abs/2005.09414.

[28] C. Prause, M. Scholten, A. Zimmermann, R. Reiners, and M. Eisenhauer, "Managing the iterative requirements process in a multi-national project using an issue tracker," Aug. 2008, pp. 151–159. DOI: 10.1109/ICGSE.2008.14.

[29] A. Hevner, S. March, J. Park, and S. Ram, "Design science in information systems research," *Management Information Systems Quarterly*, vol. 28, no. 1, pp. 75–105, Mar. 2004.

[30] E. Knauss, "Constructive master's thesis work in industry: Guidelines for applying design science research," *CoRR*, vol. abs/2012.04966, 2020. arXiv: 2012.04966. [Online]. Available: https://arxiv.org/abs/2012.04966.

[31] R. Wieringa, "Design science as nested problem solving," in *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology*, ser. DESRIST '09, Philadelphia, PA, USA: Association for Computing Machinery, 2009. [Online]. Available: doi:10.1145/1555619.1555630.

[32] L. Palinkas, S. Horwitz, C. Green, J. Wisdom, N. Duan, and K. Hoagwood, "Purposeful Sampling for Qualitative Data Collection and Analysis in Mixed Method Implementation Research.," *Adm Policy Ment Health*, vol. 42, pp. 533–544, 2015. [Online]. Available: https://doi.org/10.1007/s10488-013-0528-y.

[33] R. Frances, M. Coughlan, and P. Cronin, "Interviewing in qualitative research," *International Journal of Therapy and Rehabilitation*, vol. 16, pp. 309–314, 2009. DOI: 10.12968/ijtr.2009.16.6.42433.

[34] R. Ørngreen and K. Levinsen, "Workshops as a research methodology," *Electronic Journal of e-Learning*, vol. 15, pp. 70–81, Apr. 2017.

[35] T. Jones, M. Baxter, and V. Khanduja, "A quick guide to survey research," *The Annals of The Royal College of Surgeons of England*, vol. 95, no. 1, pp. 5–7, 2013. DOI: 10.1308/003588413X13511609956372.

[36] D. Gause and G. Weinberg, *Are your lights on?* New York, NY, USA: Winthrop, 1982, p. 15.

[37] P. Carlshamre, K. Sandahl, M. Lindvall, B. Regnell, and J. Natt och Dag, "An industrial survey of requirements interdependencies in software product release planning," pp. 84–91, 2001. DOI: 10.1109/ISRE.2001.948547.

[38] GitLab, "Introduction To GitLab Flow," n.d. Accessed on: Jan. 3, 2022. [Online]. Available: https://docs.gitlab.com/ee/topics/gitlab_flow.html.

[39] Gerrit Review, "Gerrit Code Review for Git," n.d. Accessed on: Apr. 13, 2022. [Online]. Available: https://gerrit-review.googlesource.com/Documentation/.

[40] S. Alliance, "The Scrum Team Roles and Accountabilities," n.d. Accessed on: May 25th, 2022. [Online]. Available: https://resources.scrumalliance.org/Article/scrum-team.

[41] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, pp. 131–164, 2008. DOI: 10.1007/s10664-008-9102-8.

A

Artifact

Guidelines

for the Automation of Requirements Feedback

Birgitta Feldís Bjarkadóttir - felds@student.chalmers.se
Heiðrún Valdís Heiðarsdóttir - valdis@student.chalmers.se

Contents

Introduction
Problem Overview
Structure of User Stories / Explanation
Interdependencies – *Rating*
Interdependencies – *User Stories*
Change History – *Rating*
Change History – *User Stories*
Language – *Rating*
Language – *User Stories*
Technical Feedback – *Rating*
Technical Feedback – *User Stories*
Proposed Abstraction Levels

Introduction

This document acts as guidelines as to how feedback can be introduced to requirement changes.

In agile systems, changes made to requirements are often made simultaneously by different teams using different processes and tools [1], which can introduce issues and inconsistencies of the final product, potentially leading to project failure and business loss [2]. By incorporating continuous integration methods that eliminate manual work, shorten the release cycle and improve software quality [3] to requirements, the goal of these guidelines is to help reduce inconsistencies in requirements specifications, thus leading to higher success rate of products.

The guidelines are presented as user stories, each for a different type of feedback that is intended to solve certain common problems within requirements verification and validation. The common problems of requirements verification and validation are presented in an overview that links the problems to the category they belong to.

Both the problems and user stories are divided into four categories, related to what kind of feedback the corresponding user story targets. The categories are: Interdependencies (I), Change History (C), Language (L) and Technical Feedback (T). The problems are identified with the code P<Name of category><ID>.

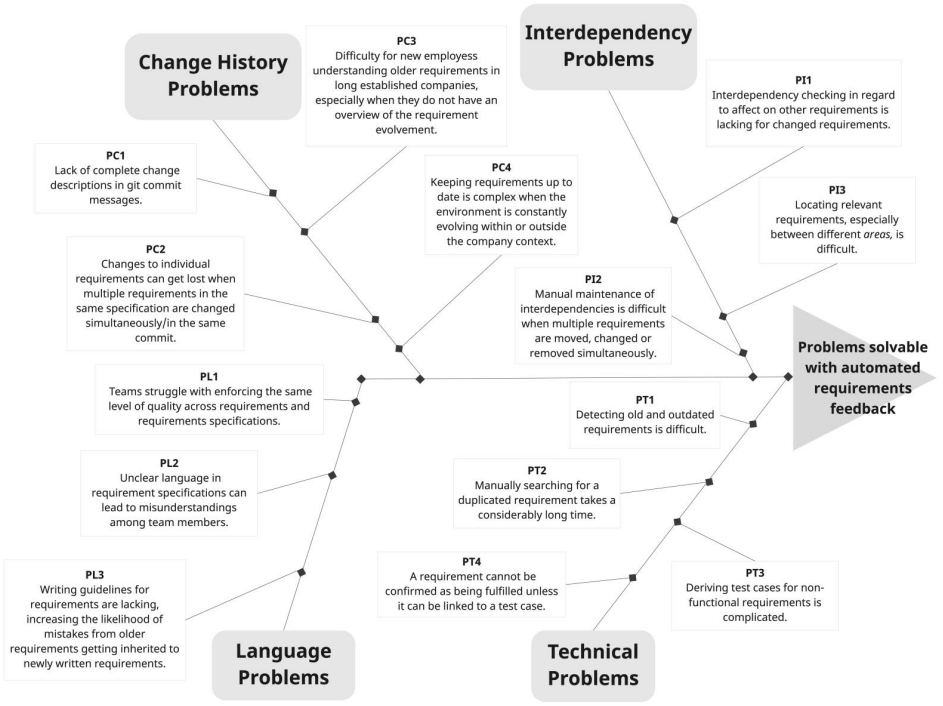
Each category section begins with an overview matrix of the business value and implementation cost of each user story, as estimated by requirements experts. Following the matrix are the user stories of the corresponding feedback category.

Each user story begins with the feedback type it targets, followed by a list of problems that the feedback type is intended to solve. In addition, the estimated business value and implementation cost of the user story are indicated on the user story card. Included on each user story slide is a table of the appropriate timings in a git-based workflow of when to give the certain feedback, as well as indicating which individuals should receive the feedback at the given timing. The receivers at each timing are ordered by which of them is more relevant for the feedback, according to how they were rated by requirements experts. The timings give an indication of where feedback can be relevant to the corresponding receiver. Frequency of the feedback, that is if it should be delivered at *all those timings*, for the same feedback type, is not taken into consideration in these guidelines.

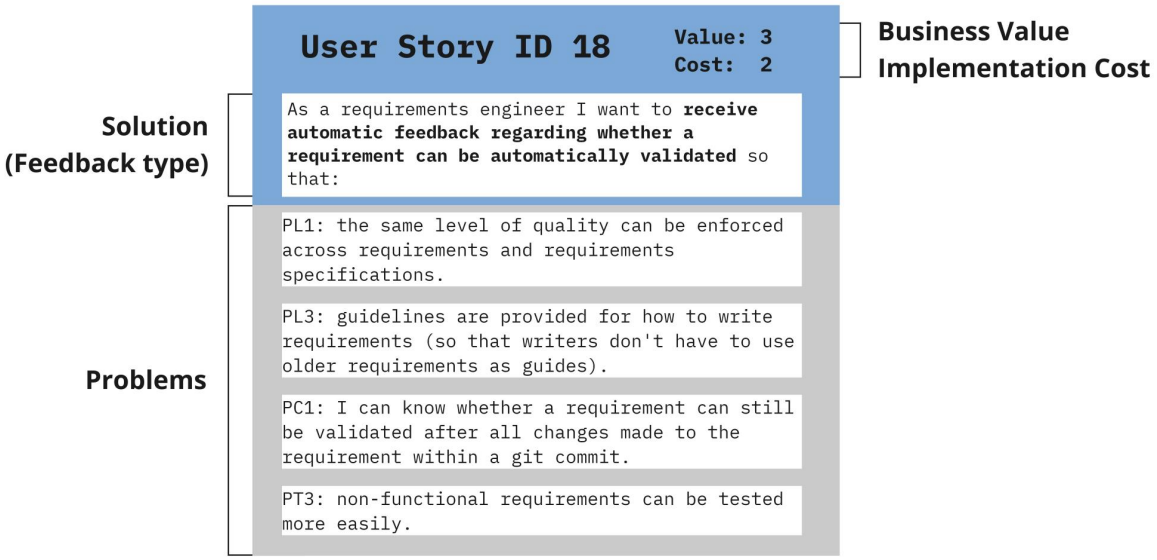
Following the user stories, a prioritized list of the feedback is presented that shows the user stories' priority, value and cost, as estimated by requirements experts. Finally, suggestions of abstraction levels to give the feedback types are proposed.

The guidelines aim to help requirement specialists determine which feedback to introduce to their own requirements management systems based on which problems exist in their context. The business value and implementation effort serve as a guide to selecting which feedback types to incorporate into their systems and how they can be prioritized. The guidelines are intended as a support to specialists in making decisions. However, it is important to stay critical and aware of the context in question.

Problem Overview



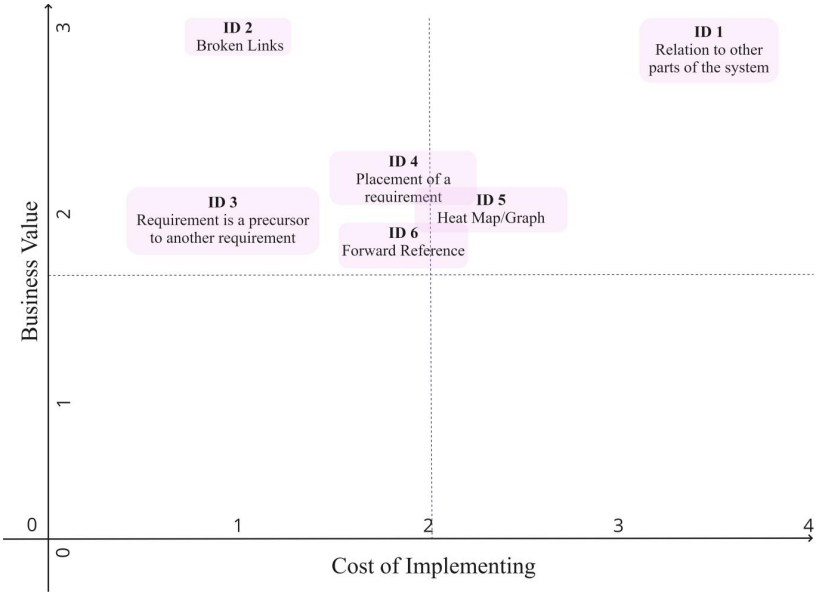
Structure of User Stories / Explanations



5

Rating of Interdependency Feedback

	Business Value (1-3)	Cost of Implementing (1-4)
ID 1	3	3.5
ID 2	3	1
ID 3	2	1
ID 4	2	2
ID 5	2	2
ID 6	2	2



6

User Story ID 1 **Value: 3** **Cost: 3.5**

As a requirements engineer I want to be able to **see relations to other parts of the system** so that:

PI1: interdependencies can be checked in relation to how changed requirements affect other requirements.

PI2: linked requirements can easily be maintained when a requirement is changed, moved or removed from within a specification or between specifications.

PI3: relevant requirement can easily be located, especially those that belong do different areas.

PC1: I can have a clear view of everything that has been changed in a requirement.

PC3: I can understand *old* requirements more easily and thereby understand the system.

PC4: requirements can easily be kept up to date in relation to the environment within the company context or outside of it.

PT1: *old* requirements, that have no dependencies to other requirements or artifacts and should likely be removed, can be detected.

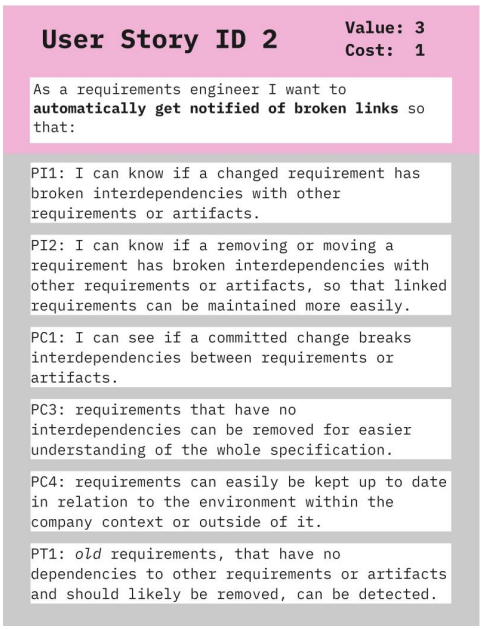
PT2: duplicated requirements can quickly be identified.

PT4: it can be ensured that the requirement is linked to a test case (and thereby can be confirmed as being fulfilled).

Timing - Individual	**Receivers**
During writing	Author Requirements responsible/owner
When reviewing	Requirements responsible/owner Author
Git commit	Author Requirements responsible/owner
Pushing	Author Requirements responsible/owner
Maintenance checks	Requirements responsible/owner
Any time	Requirements responsible/owner Author

Timing - Sets	**Receivers**
During writing	Author Requirements responsible/owner
When reviewing	Requirements responsible/owner Author
Git commit	Author Requirements responsible/owner
Pushing	Author Requirements responsible/owner
Any time	Author Requirements responsible/owner

7



Timing - Individual	Receivers
During writing	Author
When reviewing	Author Requirements responsible/owner
Git commit	Author Requirements responsible/owner
Pushing	Author Requirements responsible/owner
Maintenance checks	Requirements responsible/owner Author
Any time	Author Requirements responsible/owner

Timing - Sets	Receivers
During writing	Author Requirements responsible/owner
When reviewing	Author Requirements responsible/owner
Git commit	Author Requirements responsible/owner
Pushing	Author Requirements responsible/owner
Any time	Author Requirements responsible/owner

8

User Story ID 3 **Value: 2** **Cost: 1**

As a requirements engineer I want to **automatically be told that a requirement is a precursor to another requirement** so that:

PI1: effects of changes to the requirement on subsequent requirements can be considered when making changes to the requirement.

PI2: linked requirements can be maintained more easily since requirement changes, movements or removals are made with informed decisions based on ancestor requirements.

PI3: relevant requirement can easily be located, especially those that belong do different areas.

PC1: I know if changes to the requirement have made it a precursor to another requirement.

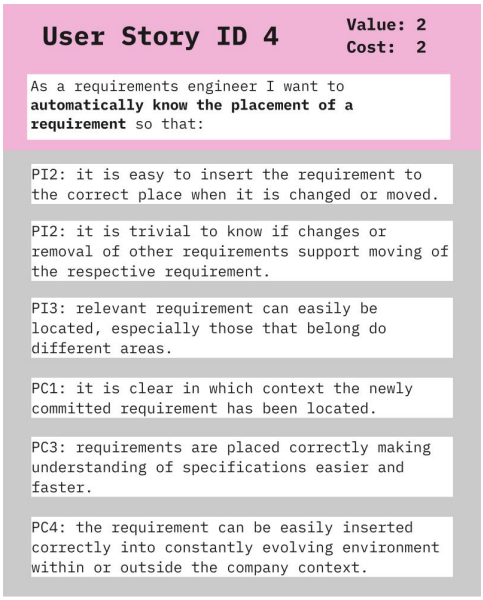
PC3: the context of requirements within a specification are easier to understand.

PC4: requirements can easily be kept up to date in relation to the environment within the company context or outside of it.

Timing - Individual	**Receivers**
During writing	Author Requirements responsible/owner
When reviewing	Author
Git commit	Author Requirements responsible/owner
Pushing	Author
Maintenance checks	
Any time	Author

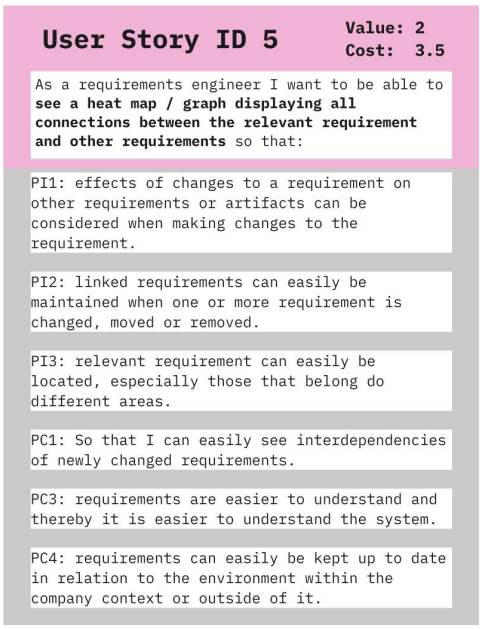
Timing - Sets	**Receivers**
During writing	Author Requirements responsible/owner
When reviewing	Author
Git commit	Author Requirements responsible/owner
Pushing	Requirements responsible/owner Author
Any time	Requirements responsible/owner Author

9



Timing - Individual	**Receivers**
During writing	Author Requirements responsible/owner
When reviewing	Author Requirements responsible/owner
Git commit	Author Requirements responsible/owner
Pushing	Author Requirements responsible/owner
Maintenance checks	Requirements responsible/owner
Any time	Author

Timing - Sets	**Receivers**
During writing	Author Requirements responsible/owner
When reviewing	Author Requirement responsible/owner
Git commit	Author Requirements responsible/owner
Pushing	Author Requirements responsible/owner
Any time	Author Requirements responsible/owner



Timing - Individual	**Receivers**
During writing	Author Requirements responsible/owner
When reviewing	Author Requirements responsible/owner
Git commit	Author Requirements responsible/owner
Pushing	Author Requirements responsible/owner
Maintenance checks	Requirements responsible / owner
Any time	Author Requirements responsible / owner

Timing - Sets	**Receivers**
During writing	Author Requirements responsible/owner
When reviewing	Requirements responsible/owner Author
Git commit	Author Requirements responsible/owner
Pushing	Requirements responsible/owner Author
Any time	Requirements responsible / owner Author

11

User Story ID 6 **Value: 2** **Cost: 2**

As a requirements engineer I want to **be notified when a functionality is used in my requirement before it is defined** so that:

PI2: linked requirements can easily be maintained when one or more of them is changed, moved or removed.

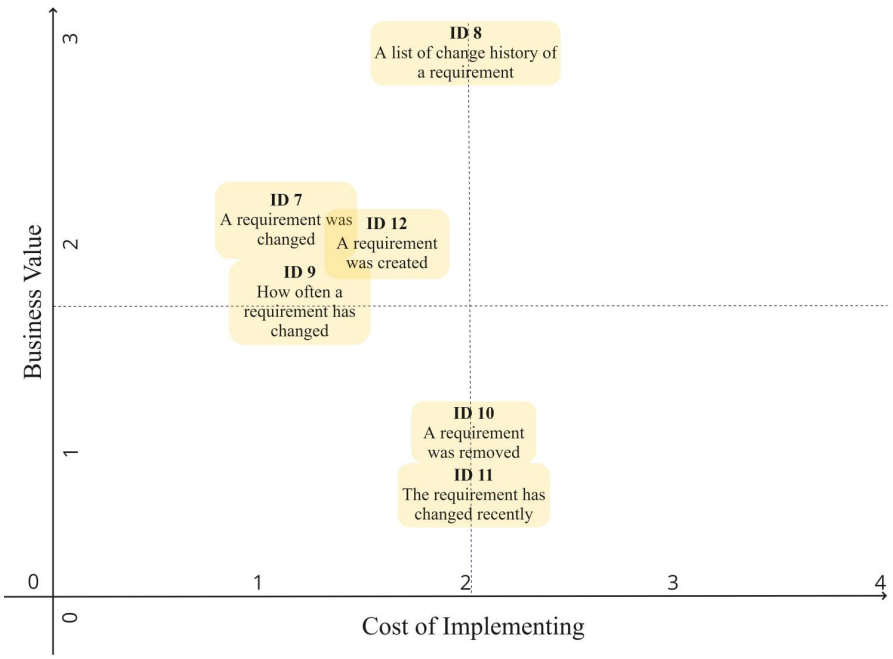
PC4: I can keep track of my requirements being up to date in a constantly evolving environment within or outside the company context.

Timing - Individual	**Receivers**
During writing	Author
When reviewing	Author
Git commit	Author
Pushing	Author
Maintenance checks	Requirements responsible/owner
Any time	Author

Timing - Sets	**Receivers**
During writing	Author
When reviewing	Author
Git commit	Author
Pushing	Author
Any time	Author Requirements responsible/owner

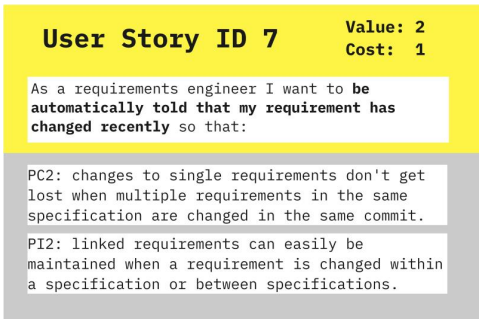
12

Rating of Change History Feedback



	Business Value (1-3)	Cost of Implementing (1-4)
ID 7	2	1
ID 8	3	2
ID 9	2	1
ID 10	1	2
ID 11	1	2
ID 12	2	1

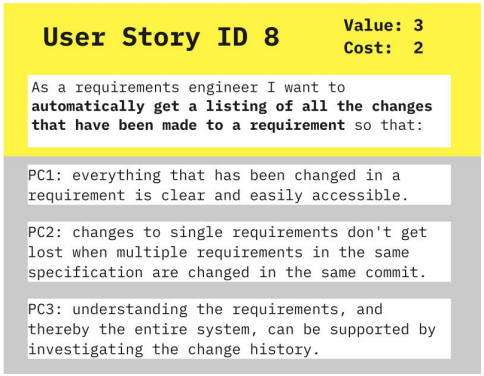
13



Timing - Individual	Receivers
When reviewing	Requirements responsible/owner Author
Any time	Requirements responsible/owner Author

Timing - Sets	Receivers
Maintenance checks	Requirements responsible/owner
Any time	Author Requirements responsible/owner

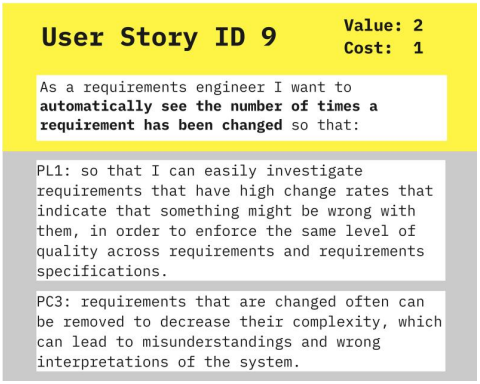
14



Timing - Individual	Receivers
When reviewing	Author Requirements responsible/owner
Maintenance checks	Requirements responsible/owner
Any time	Requirements responsible/owner Author

Timing - Sets	Receivers
Maintenance checks	Requirements responsible/owner
Any time	Author Requirements responsible/owner

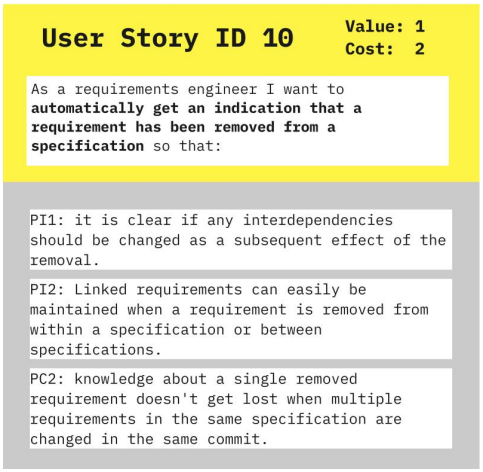
15



Timing - Individual	Receivers
When reviewing	Requirements responsible/owner Author
Maintenance checks	Requirements responsible/owner
Any time	Requirements responsible/owner

Timing - Sets	Receivers
Maintenance checks	Requirements responsible/owner
Any time	Author Requirements responsible/owner

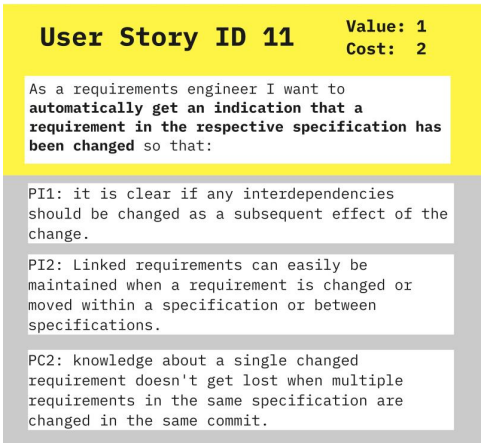
16



Timing - Individual	Receivers
When reviewing	Requirements responsible/owner Author
Maintenance checks	Requirements responsible/owner
Any time	Requirements responsible/owner Author

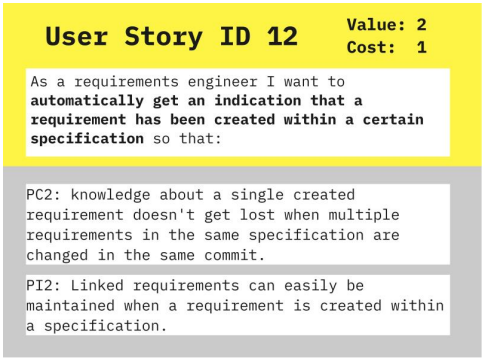
Timing - Sets	Receivers
Maintenance checks	Requirements responsible/owner
Any time	Author Requirements responsible/owner

17



Timing - Individual	Receivers
Maintenance checks	Requirements responsible / owner

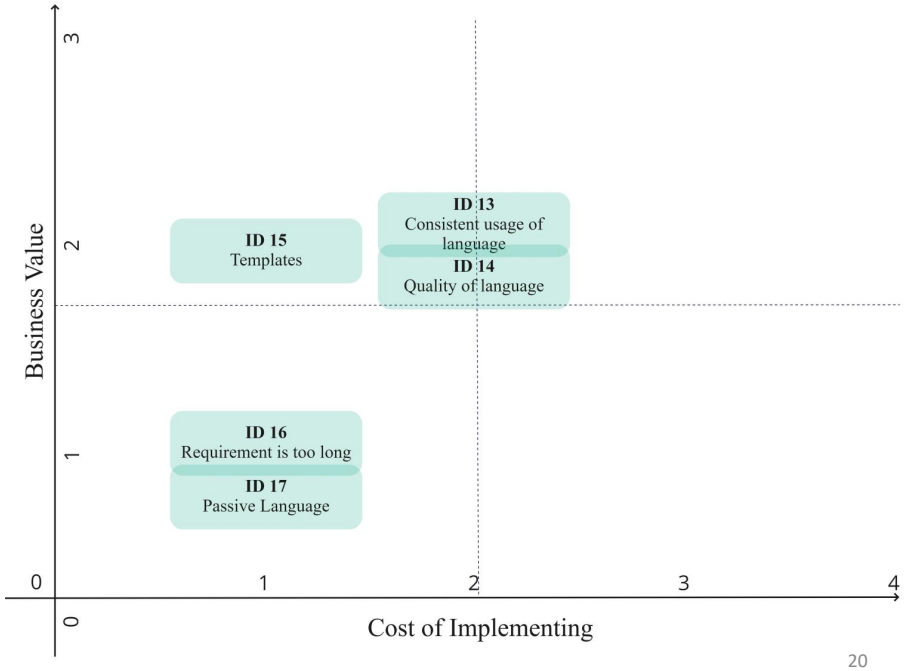
18


User Story ID 12
Value: 2
Cost: 1
As a requirements engineer I want to automatically get an indication that a requirement has been created within a certain specification so that:
PC2: knowledge about a single created requirement doesn't get lost when multiple requirements in the same specification are changed in the same commit.
PI2: Linked requirements can easily be maintained when a requirement is created within a specification.

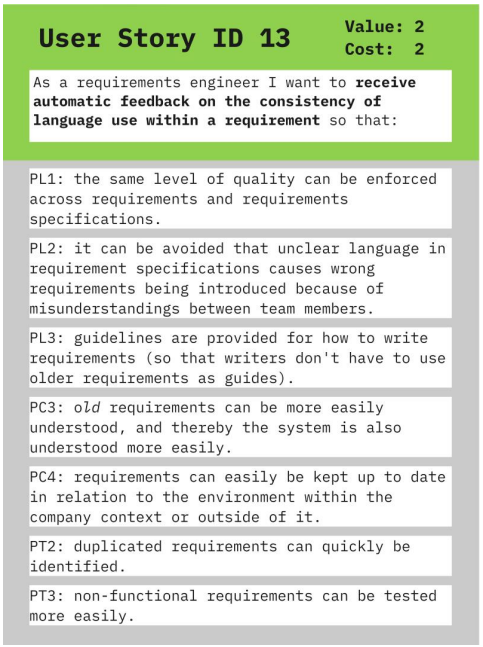
19

Rating of Language Feedback

	Business Value (1-3)	Cost of Implementing (1-4)
ID 13	2	2
ID 14	3	2
ID 15	2	1
ID 16	1	1
ID 17	1	1



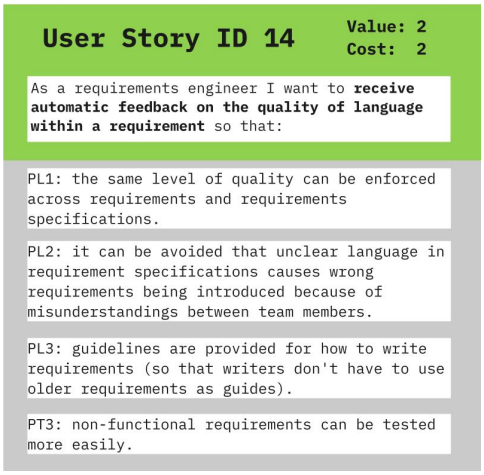
20



Timing - Individual	Receivers
During writing	Author Requirements responsible/owner
When reviewing	Author Requirements responsible / owner
Git commit	Author
Any time	Author

Timing - Sets	Receivers
Any time	Author

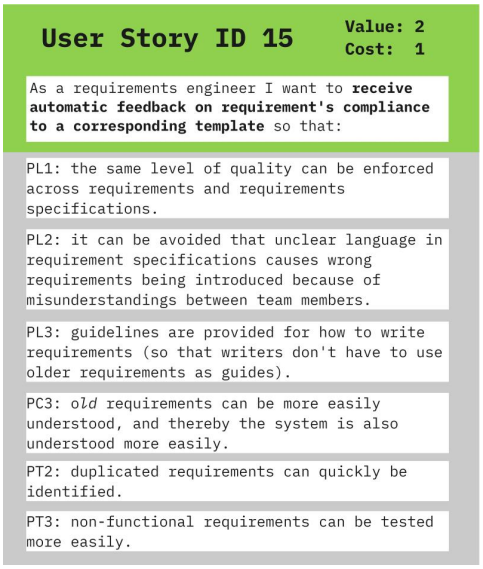
21



Timing - Individual	Receivers
During writing	Author Requirements responsible/owner
When reviewing	Author Requirements responsible/owner
Git commit	Author
Pushing	Author
Maintenance checks	Requirements responsible/owner
Any time	Author

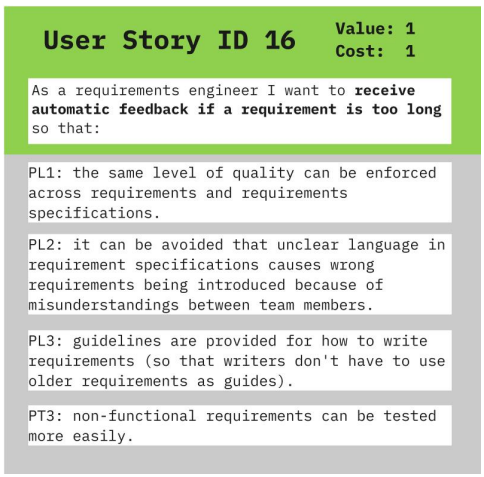
Timing - Sets	Receivers
Any time	Author Requirements responsible/owner

22



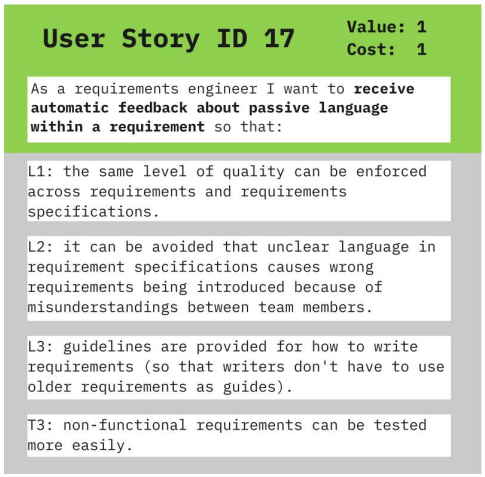
Timing - Individual	Receivers
During writing	Author
When reviewing	Author Requirements responsible/owner
Git commit	Author Requirements responsible/owner
Pushing	Author
Any time	Author

Timing - Sets	Receivers
Any time	Author



Timing - Individual	Receivers
During writing	Author
When reviewing	Author
Git commit	Author
Pushing	Author
Any time	Author

Timing - Sets	Receivers
Any time	Author

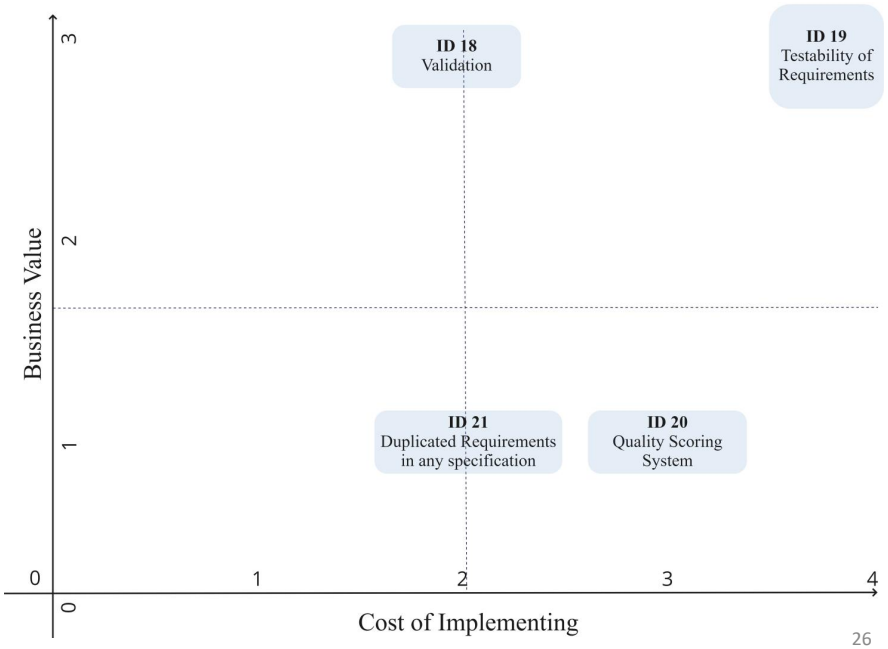


Timing - Individual	Receivers
During writing	Author
Git commit	Author
Pushing	Author
Any time	Author

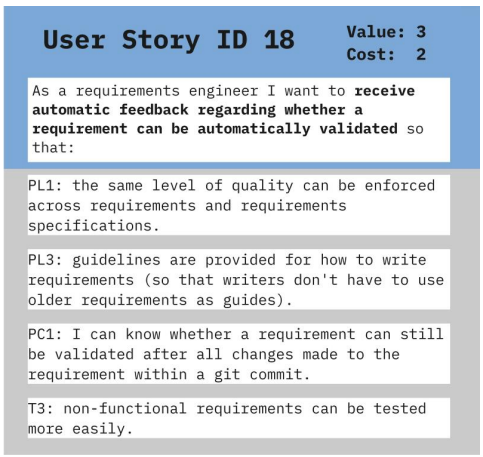
Timing - Sets	Receivers
Any time	Author

Rating of Technical Feedback

	Business Value (1-3)	Cost of Implementing (1-4)
ID 18	3	2
ID 19	3	3.5
ID 20	1	3
ID 21	1	2

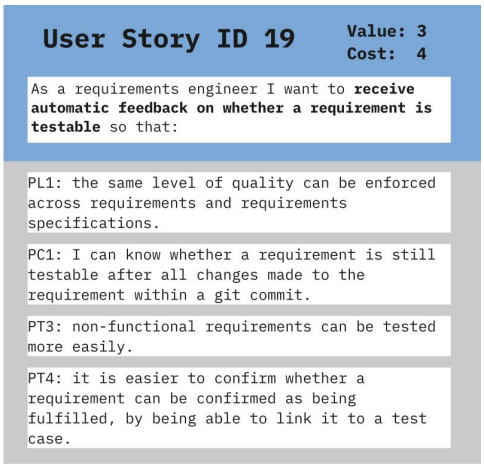


26



Timing - Individual	Receivers
When reviewing	Author
Git commit	Author
Pushing	Author
Maintenance checks	
Any time	Author Requirements responsible / owner

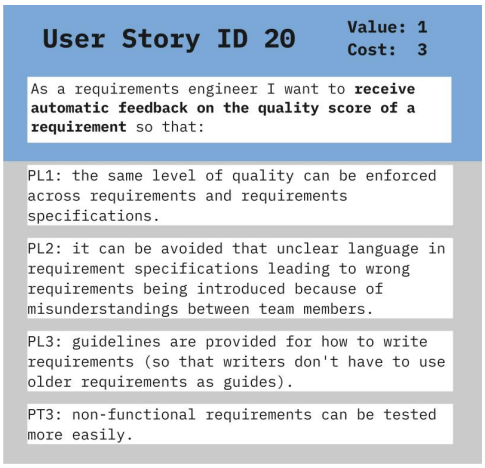
Timing - Sets	Receivers
Pushing	Author Requirements responsible / owner
Any time	Author Requirements responsible / owner



Timing - Individual	**Receivers**
When reviewing	Author
Git commit	Author Requirements responsible/owner
Pushing	Author Requirements responsible/owner
Maintenance checks	Requirements responsible/owner
Any time	Author Requirements responsible/owner

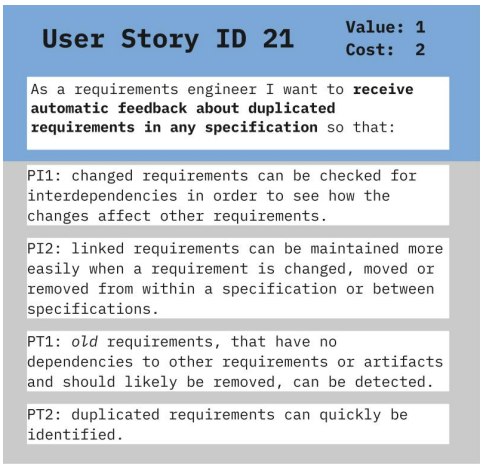
Timing - Sets	**Receivers**
Pushing	Author Requirements responsible / owner
Any time	Author Requirements responsible/owner

28



Timing - Individual	Receivers
When reviewing	Author Requirements responsible/owner
Git commit	Author Requirements responsible/owner
Pushing	Author Requirements responsible/owner
Maintenance checks	Requirements responsible/owner
Any time	Author Requirements responsible/owner

Timing - Sets	Receivers
Any time	Author Requirements responsible/owner



Timing - Individual	Receivers
When reviewing	Author Requirements responsible/owner
Git commit	Author Requirements responsible/owner
Pushing	Author Requirements responsible/owner
Maintenance checks	Requirements responsible/owner
Any time	Author Requirements responsible/owner

Timing - Sets	Receivers
During writing	Author Requirements responsible/owner
When reviewing	Author Requirements responsible/owner
Git commit	Author Requirements responsible/owner
Pushing	Author Requirements responsible/owner
Maintenance checks	Requirements responsible/owner Author
Any time	Author Requirements responsible/owner

Prioritized List of Feedback

Suggestions of the prioritization of feedback types, ordered by Business Value (1 = Low, 2 = Medium, 3 = High) and Cost of Implementing (1 = Small, 2 = Medium, 3 = Large, 4 = Huge).

User Story ID	Business Value (1-3)	Cost of Implementing (1-4)
2	3	1
8	3	2
18	3	2
1	3	3.5
19	3	4
15	2	1
9	2	1
3	2	1
7	2	1
12	2	1
13	2	2
14	2	2
6	2	2
4	2	2
5	2	3.5
16	1	1
17	1	1
10	1	2
11	1	2
21	1	2
20	1	3

31

Proposed Abstraction Levels

Suggestions are to give feedback at the abstraction level...

Error when something is completely wrong.

Errors should only be given when the changes should not be accepted into the workflow.

Errors should indicate a full stop and ban further proceeding without the reported issue being fixed.

Warning when there is something that could be wrong somehow, but is tolerated to be pushed or accepted into the workflow.

Resolving warnings should be taken into consideration, but can be ignored for valid reasons.

Information when the feedback provides instructions or advice on increased quality of requirements.

Information is something that is useful to receive while working on requirements or specifications to decrease the subsequent review time.

Information does not necessarily have to be implemented.

Scoring System for giving requirements engineers the power of deciding if/how they respond to feedback.

A score that is given on the quality of requirements, for example, language quality, similarity, criticality or severity.

High Score = High Quality
Low Score = Low Quality

Allows requirements engineers to decide the true quality level of requirements or requirements specifications while having a score as a reference.

32

Bibliography

[1] 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, vol. 172, p. 110 851, 2021. [Online]. Available: doi:10.1016/j.jss.2020.110851.

[2] S. Jayatilleke and R. Lai, "A systematic review of requirements change management," Information and Software Technology, vol. 93, Sep. 2017. doi:10.1016/j.infsof.2017.09.004.

[3] M. Shahin, M. Ali Babar, and L. Zhu, "Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices," IEEE Access, vol. PP, Mar. 2017. [Online]. Available: doi:10.1109/ACCESS.2017.2685629.

A

Interview Guide 1

A.1 Introduction

1. Present ourselves, the thesis subject and goals of the interview/workshop.
2. Consent: Do you give your permission for our discussion being recorded? We have an NDA agreement between your company and Chalmers so your personal information will not be shared further and all answers will be anonymized.

A.2 Introductory Questions

1. Can you tell us about your experience in working with requirements, e.g. requirements creation and maintenance?
   - How long have you been working with requirements management and what are your responsibilities when it comes to managing requirements?
2. What is your role within your company (which team are you a part of?) and what kind of projects are you working on? (I.e. What does your department do?)

A.3 Main Questions

A.3.1 Change History of Requirements

1. Are multiple teams or individuals responsible for changes of requirements?
   (a) How are simultaneous changes by multiple teams or team members handled?
   (b) Which tools or systems are used to handle/update requirements? Where are they stored?
   (c) Are the tools different for different types of requirements?
2. How should changes to requirements be reviewed when a new or changed requirement is being inserted into the backlog?
   (a) How is it done now?
   (b) Why does it work well/not well now?

A.3.2 Types of Requirements Feedback

1. What types of feedback are suitable for requirements (who should review, history of requirement, etc.)?
2. At which point in the requirements delivery process, i.e. from when it is created until delivery, should they be introduced?
3. Do you think that all of these feedback types are relevant/necessary during the delivery process?
4. Which level of abstraction is, in your opinion, the most valuable in requirements feedback at which point of time in your in the requirements delivery process, i.e. from when it is created until delivery?

A.3.3 Quality of Requirements

1. In your opinion, what are the challenges that arise because of lack of quality in requirements?
2. Which elements are important when it comes to the quality of requirements? Both in your opinion and how it is done formally.
   (a) Are there any exceptions to needing to fulfill them?
3. How problematic are these challenges and issues?

A.3.4 Completeness of Requirements

Completeness: A complete requirement is one that contains all details that are needed in the context and no information is left out.[1]

1. What do you consider a complete requirement and how are they verified/validated as complete?
2. What kind of issues can arise from incomplete requirements based on your experience?
3. Do you think all completeness criteria used within your work is relevant? I.e. does it affect your work if a requirement is not defined in a specified way?
4. Do all requirements in the backlog fulfill the criteria of being complete? Why/why not?

A.3.5 Interdependencies between Requirements

1. How are dependencies between different teams handled?
2. What are the challenges that arise in relation to interdependencies of requirements?
   (a) within teams?
   (b) between different teams?
3. How are interdependencies between requirements verified/validated? Is it different when they are being updated and not created?

[1] D. Zowghi and V. Gervasi, "On the interplay between consistency, completeness, and correctness in requirements evolution," *Information and Software Technology*, vol. 45, no. 14, pp. 993–1009, 2003, Eighth International Workshop on Requirements Engineering: Foundation for Software Quality, ISSN: 0950-5849. DOI: https://doi.org/10.1016/S0950-5849(03)00100-9.

A.3.6 Consistency of Requirements

Consistency: Internal consistency. A system requirement specification is internally consistent if, and only if, no subset of individual requirements described in it conflict.[2]

1. What issues can derive from lack of consistencies in requirements?
2. Is there a validation/verification process of the consistency of requirements that have been created? That have been updated?

A.3.7 Prioritization of Requirements

1. What are the challenges that arise in relation to prioritization of requirements?
   (a) within teams?
   (b) between different teams?

A.3.8 Prototype Questions

1. Change History:
   (a) Do you think the following types of feedback would be beneficial to the process? Explain why.
   (b) Do you have any other ideas in mind regarding feedback on a requirement's change history? (Is there anything you have experienced as lacking while working with requirements?)
2. Timings of Feedback:
   (a) What is your opinion on giving feedback at the following points in the development cycle:
3. Interdependencies:
   (a) What do you think of these interdependency issue mitigation ideas?
   (b) Do you have any further suggestions for other?
4. Language processing:
   (a) What do you think of these examples of validation regarding language?
5. Technical feedback:
   (a) What do you think of these validation examples and do you have any other ideas for automatically validating requirements?
6. Types of Feedback:
   (a) What are your thoughts on giving feedback on individual requirements based on:
   (b) What are your thoughts on giving feedback on sets of requirements based on:
   (c) Do you think some of them are more important than others?
7. Who should receive the feedback?

[2] "IEEE recommended practice for software requirements specifications," *IEEE Std 830-1998*, pp. 1-40, 1998. DOI: 10.1109/IEEESTD.1998.88286.

A.4 Extras

1. Which version control processes (e.g GitLab flow, gerrit, git flow) do you/your company use and how are they used?

A.5 Conclusion

1. Is there anything we should have asked but did not?
2. Can we follow up on your answers?
3. Is there anyone you think we would benefit from interviewing?
4. Would you be willing to participate in the next round/phase of interviews?
5. Thank you.

B

Interview Guide 2

B.1 Introduction

1. Present ourselves, the thesis subject and goals of the interview/workshop.
   Thanks for participating in this study. We are software engineering students at Chalmers doing our master thesis on Automating Requirements Feedback. The purpose of this interview is to validate the findings so far and get your expert input for further development.
   The questions will be split into a few categories, and each category has a corresponding slide in a PowerPoint that we will show you while asking the questions.
2. Consent: We have an NDA agreement between your company and Chalmers which means that your personal information will not be shared further and all answers will be anonymized.
   Do we have your permission to record the interview?

B.2 Introductory Questions

1. What is your role within your company (which team are you a part of?) and what kind of projects are you working on? (I.e. What does your department do?)
2. Can you tell us about your experience in working with requirements, e.g. requirements creation and maintenance?
   (a) How long have you been working with requirements management and what are your responsibilities when it comes to managing requirements?

B.3 Main Questions

B.3.1 Receivers of Feedback:

1. Who should automatically receive feedback regarding...
   (a) New requirements?
   (b) Changed requirements?
   (c) Removed requirements?
2. Which feedback should the author of a requirement automatically receive?
3. Which feedback should the responsible of a requirement automatically receive?

   (a) Who is usually the requirement responsible?
   (b) When working with requirements, why would it be valuable for you to automatically get information of who is the requirements responsible?
4. How valuable is it for the entire development team to receive the feedback automatically?
   (a) If yes, which feedback should they receive?
   (b) Should everyone in the team be suggested as a reviewer?
5. Do you think it is important that testers are notified of requirements changes or when new requirements are introduced and what kind of feedback should testers receive?

B.3.2 Change History:

1. Why would it be valuable to see what has changed in a requirement?
   (a) Would it be useful for you to know how often the requirement has been changed?
   (b) Would it be useful for you to get a full list of changes that have been made to a requirement?
2. Would it be useful for you to get contact information of the person who had changed the requirement last?
3. Would it be of value to see if a certain person has reviewed the requirement?
   (a) Only your requirement or any requirements?
4. Which (other) elements would you want to see/get warnings about when it comes to change history?
5. Are multiple teams or individuals responsible for changes of requirements?

B.3.3 Interdependencies:

1. How important is getting feedback on the interdependencies of requirements?
2. What kind of feedback would you like to receive for interdependencies?
   (a) Requirement you linked to does not exist.
   (b) The requirement you are removing is linked to another requirement.
      i. Would you want to toggle any on/off?
   (c) What is your opinion on traceability to safety/security critical requirements?
   (d) Would you want to be notified when a requirement includes forward referencing and when would that be useful?
      - *Forward referencing is when a feature is used as part of a requirement before it, itself, is defined in a requirement.*[1]
3. Would you want to get feedback on overall relations within your system?
   (a) Within your requirements specification only?
   (b) With the relevant test cases?

[1] O. Ormandjieva, I. Hussain, and L. Kosseim, "Toward a text classification system for the quality assessment of software requirements written in natural language," in *Fourth International Workshop on Software Quality Assurance: In Conjunction with the 6th ESEC/FSE Joint Meeting*, New York, NY, USA: Association for Computing Machinery, 2007, pp. 39–45. DOI: 10.1145/1295074.1295082.

4. In your opinion, how should the interdependencies be stored and managed?

B.3.4 Language:

1. How important is getting feedback on the language of requirements?
2. What feedback would you like to receive regarding the language of your requirements?
   (a) Length of the requirement? How?
      i. Max value? Split at certain words?
3. How would it help you to have definitions of specific terms for consistent usage of language throughout your requirements specification? (To avoid duplicates and inconsistencies.)
   (a) How about within the entire company?
   (b) Who should define the language standard?
4. How important is getting feedback on passive language in requirements?
5. How would it help you to have templates available, that can automatically be compared to the requirement you are working on to ensure consistencies in the ways requirements are written?
   (a) Where in your flow of work do you imagine that templates for requirements would fit in?
6. Which of these would you want to be on the form of errors and which of these would you want to be on the form of information or warning?

B.3.5 Technical Feedback:

1. How useful is it to know automatically if the same requirement has already been defined?
   (a) ...within your requirement specification?
   (b) ...within the entire department/company?
   (c) Can you think of a way to automate it?
2. How useful would it be to get feedback indicating whether it is possible to validate your requirement automatically?
   - *Validation of requirements is the process of checking whether requirements defined for development, actually describe the customer's needs.* [2]

B.3.6 Feedback for...

1. How fitting do you think the following automatic checks are for individual requirements?
   (a) Are there other feedback types for individual requirements you can think of?
2. How fitting do you think the following automatic feedback checks are for sets of requirements?

[2] H. Anas, M. Ilyas, Q. Tariq and M. Hummayun,, "Requirements validation techniques: An empirical study," *International Journal of Computer Applications*, vol. 148, pp. 5–10, 2016. DOI: 10.5120/ijca2016910911.

(a) Are there other types of sets of requirements you can think of?

B.3.7 Timings of Feedback:

1. — MIRO —
   (a) What kinds of feedback do you think are reasonable at each point in time?
   (b) What feedback would you like to receive automatically in real time during writing of requirements?
   (c) What feedback would you like to receive automatically when reviewing someone else's changes?
   (d) When pushing changes to the code base?
      i. Where should the feedback be located (In the pipeline (in Gerrit[3], GitLab[4], or command line tool (T-Reqs[5])).
   (e) When commiting to the repository?
      i. Where? (In gerrit, gitlab, or command line tool f.x T-Reqs)
2. Would you like to be notified of unresolved feedback when pushing changes?
3. Is there any other feedback you would like to receive and at which point in time during the development process?

B.3.8 Abstraction Levels:

1. When would you like feedback that you receive on requirements to be on the abstraction level of hard feedback, such as errors that cannot be ignored?
2. What do you consider the difference between a warning and information when thinking of requirements feedback? (Is there a difference?)
3. When would you like the automatic feedback to be on the form of information?
   - Alternative: Fetching with command
4. When would you like the automatic feedback to be on the form of warnings?
   - Alternative: Fetching with command
5. Where should information be displayed? (In the version control provider's workflow such as Gerrit or GitLab, or simply in the command-line tool (T-Reqs))
6. Can you think of any other classification levels?

B.3.9 Additional Questions:

1. What are the biggest challenges you encounter when working with requirements?

B.4 Conclusion

1. Is there anything you expected us to ask that we did not touch upon?

[3] `https://www.gerritcodereview.com`
[4] `https://about.gitlab.com`
[5] `https://gitlab.com/treqs-on-git/treqs-ng`

2. Can we follow up on your answers?
3. Would you be willing to participate in the next round/phase of interviews?
4. Is there anyone you think we would benefit from interviewing?
5. Thank you.

C

Survey

Automating Feedback for Requirement Changes

We are two master's students in Software Engineering at Chalmers, Birgitta Feldís Bjarkadóttir and Heiðrún Valdís Heiðarsdóttir, currently working on our thesis Automating Feedback for Requirement Changes.

We are investigating which types of feedback should be automated to make the requirements engineering process easier and quicker, resulting in a faster and less expensive process.

There is a trend to use version control systems for requirements, such as git, and a part of the investigation is to examine at which time in the development process/git workflow to give each type of **automated** feedback and would highly appreciate to get your expert opinion on those matters by answering our survey. The length of the survey is approximately 20 minutes and includes 4 introductory and 11 main questions where you will be asked to select which roles should receive automated feedback, given a certain time in the development process/git workflow and a given feedback type. Note that the order of items listed is not supposed to indicate any priority.

Please read the instructions for every question carefully. The first part of the survey includes 4 questions that are aimed to help us in putting the answers into context through knowledge of the demographic. The questions will refer to either **individual requirements** or **sets of requirements**, where individual requirements refers to the automated feedback that is being given on a single requirement and sets of requirements refers to the automated feedback that is being given on a collection of requirements such as a requirements specification. Each question's header will indicate which type of requirements is being referred to, as well as giving the timing of said feedback. The last question only refers to sets of requirements.

All answers are anonymous and can not be traced back to the respondent. The data will be stored and made accessible to the two researchers, supervisor and examiner.

This survey is scheduled to accept responses until (and including) Thursday, May 19th.

Do not hesitate to contact us if you have any questions regarding the study and/or specific questions in the survey.

With kind regards,
Birgitta Feldís Bjarkadóttir (felds@student.chalmers.se)
Heiðrún Valdís Heiðarsdóttir (valdis@student.chalmers.se)

◯ I hereby consent that my answers will be used for research purposes.

Page 02

Background > During Writing > On Commit > On Push > When Reviewing > During Maintenance > Any Time

Background

Please provide information regarding your background when it comes to working with requirements. Knowing the context of the responses will help interpreting the results.

1. Which description is the best fit for the industry you are involved in at the place of your employment?

- ◯ Automotive industry
- ◯ Software industry
- ◯ Academia/Research
- ◯ Other (Please specify)
- ◯ Prefer not to answer

2. Which role/s fit you most accurately in your line of work when it comes to requirements management?

- ☐ Author of requirement
- ☐ Part of development team
- ☐ Test engineer
- ☐ Requirements responsible
- ☐ Requirements owner
- ☐ Product owner
- ☐ Academic researcher
- ☐ Other (Please specify)

3. What is your work experience, measured in years, in your current line of work?

- ◯ Less than a year
- ◯ 1-2 years
- ◯ 2-4 years
- ◯ 4-6 years
- ◯ More than 6 years
- ◯ I have not worked with requirements at all
- ◯ Prefer not to answer

4. How long have you worked with requirements?

- ◯ Less than a year
- ◯ 1-2 years
- ◯ 2-4 years
- ◯ 4-6 years
- ◯ More than 6 years
- ◯ I have not worked with requirements at all
- ◯ Prefer not to answer

Page 03

Background > **During Writing** > On Commit > On Push > When Reviewing > During Maintenance > Any Time

Individual vs. Sets of Requirements

Individual requirements: Automated feedback is being given within the scope of a single requirement.

Sets of requirements: Automated feedback is given within the scope of a collection of requirements, such as a requirements specification that includes multiple individual requirements, or any other group of requirements.

During Writing of Requirements

This page involves questions on who you think should automatically receive feedback during writing of a requirement when the requirement is being created or changed.

5. During Writing of Requirement – Individual Requirements

Please select every person/role who you think should receive the corresponding type of automated feedback about a single requirement.

Automated feedback regarding...	Author of requirement	Test engineers	Requirements responsible / owner	Development team	Product owner	I don't know
Consistent usage of language	☐	☐	☐	☐	☐	☐
Quality of language	☐	☐	☐	☐	☐	☐
Adherence to template	☐	☐	☐	☐	☐	☐
Placement of requirements – Requirement might be in the wrong specification	☐	☐	☐	☐	☐	☐
Requirement is too long	☐	☐	☐	☐	☐	☐
Relation to other parts of the system	☐	☐	☐	☐	☐	☐
Forward Reference – definitions are in the wrong order	☐	☐	☐	☐	☐	☐
Passive language	☐	☐	☐	☐	☐	☐
Broken links	☐	☐	☐	☐	☐	☐
Requirement is a precursor to another requirement	☐	☐	☐	☐	☐	☐
Heat map / Graph of all connected requirements	☐	☐	☐	☐	☐	☐

6. During Writing of Requirements – Sets of Requirements

Please select every person/role who you think should receive the corresponding type of automated feedback about a set of requirements.

Automated feedback regarding...	Author of requirement	Test engineers	Requirements responsible / owner	Development team	Product owner	I don't know
Relation to other parts of the system	☐	☐	☐	☐	☐	☐
Broken links	☐	☐	☐	☐	☐	☐
Heat map / Graph of all connected requirements	☐	☐	☐	☐	☐	☐
Requirement is a precursor to another requirement	☐	☐	☐	☐	☐	☐
Forward reference – definitions are in the wrong order	☐	☐	☐	☐	☐	☐
Placement of requirements – Requirements might be in the wrong specification	☐	☐	☐	☐	☐	☐
Duplicated requirements	☐	☐	☐	☐	☐	☐

Page 04

Background > During Writing > **On Commit** > On Push > When Reviewing > During Maintenance > Any Time

Individual vs. Sets of Requirements

Individual requirements: Automated feedback is being given within the scope of a single requirement.

Sets of requirements: Automated feedback is given within the scope of a collection of requirements, such as a requirements specification that includes multiple individual requirements, or any other group of requirements.

When Committing Requirement Changes

This page involves questions on who you think should automatically receive feedback when a requirement change is committed to a git-based process.

7. When Making Commits to a Git-based Process – Individual Requirements

Please select every person/role who you think should receive the corresponding type of automated feedback about a single requirement.

Automated feedback regarding...	Author of requirement	Test engineers	Requirements responsible / owner	Development team	Product owner	I don't know
Broken links	☐	☐	☐	☐	☐	☐
Relation to other parts of the system	☐	☐	☐	☐	☐	☐
Heat map / Graph of all connected requirements	☐	☐	☐	☐	☐	☐
Placement of requirements – Requirement might be in the wrong specification	☐	☐	☐	☐	☐	☐
Forward reference – definitions are in the wrong order	☐	☐	☐	☐	☐	☐
Requirement is a precursor to another requirement	☐	☐	☐	☐	☐	☐
Passive language	☐	☐	☐	☐	☐	☐
Quality of language	☐	☐	☐	☐	☐	☐
Adherence to template	☐	☐	☐	☐	☐	☐
Consistent usage of language	☐	☐	☐	☐	☐	☐
Requirement is too long	☐	☐	☐	☐	☐	☐
Duplicated requirements	☐	☐	☐	☐	☐	☐
Quality scoring system	☐	☐	☐	☐	☐	☐
Validation	☐	☐	☐	☐	☐	☐
Testability of requirements	☐	☐	☐	☐	☐	☐

8. When Making Commits to a Git-based Process – Sets of Requirements

Please select every person/role who you think should receive the corresponding type of automated feedback about a set of requirements.

Automated feedback regarding...	Author of requirement	Test engineers	Requirements responsible / owner	Development team	Product owner	I don't know
Relation to other parts of the system	☐	☐	☐	☐	☐	☐
Broken links	☐	☐	☐	☐	☐	☐
Heat map / Graph of all connected requirements	☐	☐	☐	☐	☐	☐
Requirement is a precursor to another requirement	☐	☐	☐	☐	☐	☐
Forward reference – definitions are in the wrong order	☐	☐	☐	☐	☐	☐
Placement of requirements – Requirements might be in the wrong specification	☐	☐	☐	☐	☐	☐
Duplicated requirements in the same specification	☐	☐	☐	☐	☐	☐
Duplicated requirements in any specification	☐	☐	☐	☐	☐	☐

Page 05

Background > During Writing > On Commit > **On Push** > When Reviewing > During Maintenance > Any Time

Individual vs. Sets of Requirements

Individual requirements: Automated feedback is being given within the scope of a single requirement.

Sets of requirements: Automated feedback is given within the scope of a collection of requirements, such as a requirements specification that includes multiple individual requirements, or any other group of requirements.

When Pushing Changes

This page involves questions on who you think should automatically receive feedback when requirement changes are being pushed to a code-base or repostitory.

9. When Pushing Changes to Code-base/Repository – Individual Requirements

Please select every person/role who you think should receive the corresponding type of automated feedback about a single requirement.

Automated feedback regarding...	Author of requirement	Test engineers	Requirements responsible / owner	Development team	Product owner	I don't know
Broken links	☐	☐	☐	☐	☐	☐
Relation to other parts of the system	☐	☐	☐	☐	☐	☐
Heat map / Graph of all connected requirements	☐	☐	☐	☐	☐	☐
Placement of requirements – Requirement might be in the wrong specification	☐	☐	☐	☐	☐	☐
Forward reference – definitions are in the wrong order	☐	☐	☐	☐	☐	☐
Requirement is a precursor to another requirement	☐	☐	☐	☐	☐	☐
Passive language	☐	☐	☐	☐	☐	☐
Quality of language	☐	☐	☐	☐	☐	☐
Adherence to template	☐	☐	☐	☐	☐	☐
Consistenct usage of language	☐	☐	☐	☐	☐	☐
Requirement is too long	☐	☐	☐	☐	☐	☐
Duplicated requirements	☐	☐	☐	☐	☐	☐
Quality scoring system	☐	☐	☐	☐	☐	☐
Validation	☐	☐	☐	☐	☐	☐
Testability of requirements	☐	☐	☐	☐	☐	☐

10. When Pushing Changes to Code-base/Repository – Sets of Requirements

Please select every person/role who you think should receive the corresponding type of automated feedback about a set of requirements.

Automated feedback regarding...	Author of requirement	Test engineers	Requirements responsible / owner	Development team	Product owner	I don't know
Relation to other parts of the system	☐	☐	☐	☐	☐	☐
Broken links	☐	☐	☐	☐	☐	☐
Heat map / Graph of all connected requirements	☐	☐	☐	☐	☐	☐
Requirement is a precursor to another requirement	☐	☐	☐	☐	☐	☐
Forward reference – definitions are in the wrong order	☐	☐	☐	☐	☐	☐
Placement of requirements – Requirements might be in the wrong specification	☐	☐	☐	☐	☐	☐
Duplicated requirements in the same specification	☐	☐	☐	☐	☐	☐
Duplicated requirements in any specification	☐	☐	☐	☐	☐	☐
Quality scoring system	☐	☐	☐	☐	☐	☐
Validation	☐	☐	☐	☐	☐	☐
Testability of requirements	☐	☐	☐	☐	☐	☐

Page 06

Background > During Writing > On Commit > On Push > **When Reviewing** > During Maintenance > Any Time

Individual vs. Sets of Requirements

Individual requirements: Automated feedback is being given within the scope of a single requirement.

Sets of requirements: Automated feedback is given within the scope of a collection of requirements, such as a requirements specification that includes multiple individual requirements, or any other group of requirements.

When Reviewing Requirement Changes

This page involves questions on who you think should automatically receive feedback when a committed requirement change is reviewed.

11. When Reviewing Someone Else's Requirement Changes – Individual Requirements

Please select every person/role who you think should receive the corresponding type of automated feedback about a single requirement.

Automated feedback regarding...	Author of requirement	Test engineers	Requirements responsible / owner	Development team	Product owner	I don't know
Consistent usage of language	☐	☐	☐	☐	☐	☐
Adherence to template	☐	☐	☐	☐	☐	☐
Requirement is too long	☐	☐	☐	☐	☐	☐
A list of the change history of a requirement	☐	☐	☐	☐	☐	☐
The requirement was changed	☐	☐	☐	☐	☐	☐
How often a requirement has changed	☐	☐	☐	☐	☐	☐
A requirement was removed	☐	☐	☐	☐	☐	☐
Duplicated requirements	☐	☐	☐	☐	☐	☐
Quality scoring system	☐	☐	☐	☐	☐	☐
Validation	☐	☐	☐	☐	☐	☐
Testability of requirements	☐	☐	☐	☐	☐	☐
Broken links	☐	☐	☐	☐	☐	☐
Relation to other parts of the system	☐	☐	☐	☐	☐	☐
Heat map / Graph of all connected requirements	☐	☐	☐	☐	☐	☐
Placement of requirements – Requirement might be in the wrong specification	☐	☐	☐	☐	☐	☐
Forward reference – definitions are in the wrong order	☐	☐	☐	☐	☐	☐
Requirement is a precursor to another requirement	☐	☐	☐	☐	☐	☐

12. When Reviewing Someone Else's Requirement Changes – Sets of Requirements

Please select every person/role who you think should receive the corresponding type of automated feedback about a set of requirements.

Automated feedback regarding...	Author of requirement	Test engineers	Requirements responsible / owner	Development team	Product owner	I don't know
Relation to other parts of the system	☐	☐	☐	☐	☐	☐
Broken links	☐	☐	☐	☐	☐	☐
Heat map / Graph of all connected requirements	☐	☐	☐	☐	☐	☐
Requirement is a precursor to another requirement	☐	☐	☐	☐	☐	☐
Forward reference – definitions are in the wrong order	☐	☐	☐	☐	☐	☐
Placement of requirements – Requirements might be in the wrong specification	☐	☐	☐	☐	☐	☐
Duplicated requirements	☐	☐	☐	☐	☐	☐

Page 07

Background > During Writing > On Commit > On Push > When Reviewing > **During Maintenance** > Any Time

Individual vs. Sets of Requirements

Individual requirements: Automated feedback is being given within the scope of a single requirement.

Sets of requirements: Automated feedback is given within the scope of a collection of requirements, such as a requirements specification that includes multiple individual requirements, or any other group of requirements.

During Automatic Maintenance Checks

This page involves questions on who you think should automatically receive feedback when automatic maintenance check are run on existing requirements.

13. During Automatic Maintenance Checks – Individual Requirements

Please select every person/role who you think should receive the corresponding type of automated feedback about a single requirement.

Automated feedback regarding...	Author of requirement	Test engineers	Requirements responsible / owner	Development team	Product owner	I don't know
Broken links	☐	☐	☐	☐	☐	☐
Relation to other parts of the system	☐	☐	☐	☐	☐	☐
Heat map / Graph of all connected requirements	☐	☐	☐	☐	☐	☐
Placement of requirements – Requirement might be in the wrong specification	☐	☐	☐	☐	☐	☐
Forward reference – definitions are in the wrong order	☐	☐	☐	☐	☐	☐
Requirement is a precursor to another requirement	☐	☐	☐	☐	☐	☐
Duplicated requirements in the same specification	☐	☐	☐	☐	☐	☐
Duplicated requirements in any specification	☐	☐	☐	☐	☐	☐
Quality scoring system	☐	☐	☐	☐	☐	☐
Validation	☐	☐	☐	☐	☐	☐
Testability of requirements	☐	☐	☐	☐	☐	☐
This requirement has changed recently	☐	☐	☐	☐	☐	☐
A list of the change history of a requirement	☐	☐	☐	☐	☐	☐
The requirement was removed	☐	☐	☐	☐	☐	☐
How often a requirement has changed	☐	☐	☐	☐	☐	☐

14. During Automatic Maintenance Checks – Sets of Requirements

Please select every person/role who you think should receive the corresponding type of automated feedback about a set of requirements.

Automated feedback regarding...	Author of requirement	Test engineers	Requirements responsible / owner	Development team	Product owner	I don't know
A requirement was changed	☐	☐	☐	☐	☐	☐
A list of the change history of a requirement	☐	☐	☐	☐	☐	☐
How often a requirement has changed	☐	☐	☐	☐	☐	☐
A requirement was removed	☐	☐	☐	☐	☐	☐
Duplicated requirements in the same specification	☐	☐	☐	☐	☐	☐
Duplicated requirements in any specification	☐	☐	☐	☐	☐	☐

Page 08

Background > During Writing > On Commit > On Push > When Reviewing > During Maintenance > **Any Time**

Individual vs. Sets of Requirements

Individual requirements: Automated feedback is being given within the scope of a single requirement.

Sets of requirements: Automated feedback is given within the scope of a collection of requirements, such as a requirements specification that includes multiple individual requirements, or any other group of requirements.

Any Time

This page involves questions on who you think should be able to access feedback at any time in the requirements creation and delivery process.

15. Accessible at Any Time – Individual Requirements

Please select every person/role who you think should receive the corresponding type of feedback about an individual requirement.

Automated feedback regarding...	Author of requirement	Test engineers	Requirements responsible / owner	Development team	Product owner	I don't know
Relation to other parts of the system	☐	☐	☐	☐	☐	☐
Broken links	☐	☐	☐	☐	☐	☐
Heat map / Graph of all connected requirements	☐	☐	☐	☐	☐	☐
Requirement is a precursor to another requirement	☐	☐	☐	☐	☐	☐
Forward reference – definitions are in the wrong order	☐	☐	☐	☐	☐	☐
Placement of requirements – Requirements might be in the wrong specification	☐	☐	☐	☐	☐	☐
A list of the change history of the requirement	☐	☐	☐	☐	☐	☐
How often the requirement has changed	☐	☐	☐	☐	☐	☐
The requirement was changed	☐	☐	☐	☐	☐	☐
The requirement was removed	☐	☐	☐	☐	☐	☐
Quality of language	☐	☐	☐	☐	☐	☐
Passive language	☐	☐	☐	☐	☐	☐
Consistent usage of language	☐	☐	☐	☐	☐	☐
Requirement is too long	☐	☐	☐	☐	☐	☐
Adherence to templates	☐	☐	☐	☐	☐	☐
Validation	☐	☐	☐	☐	☐	☐
Testability of requirements	☐	☐	☐	☐	☐	☐
Quality scoring system	☐	☐	☐	☐	☐	☐

Duplicated requirements in the same specification	☐	☐	☐	☐	☐	☐
Duplicated requirements in any specification	☐	☐	☐	☐	☐	☐

16. Accessible at Any Time – Sets of Requirements

Please select every person/role who you think should receive the corresponding type of feedback about a set of requirements.

Automated feedback regarding...	Author of requirement	Test engineers	Requirements responsible / owner	Development team	Product owner	I don't know
Relation to other parts of the system	☐	☐	☐	☐	☐	☐
Broken links	☐	☐	☐	☐	☐	☐
Heat map / Graph of all connected requirements	☐	☐	☐	☐	☐	☐
Requirement is a precursor to another requirement	☐	☐	☐	☐	☐	☐
Forward reference – definitions are in the wrong order	☐	☐	☐	☐	☐	☐
Placement of requirements – Requirements might be in the wrong specification	☐	☐	☐	☐	☐	☐
A list of the change history of a requirement	☐	☐	☐	☐	☐	☐
How often a requirement has changed	☐	☐	☐	☐	☐	☐
A requirement was changed	☐	☐	☐	☐	☐	☐
A requirement was removed	☐	☐	☐	☐	☐	☐
Quality of language	☐	☐	☐	☐	☐	☐
Passive language	☐	☐	☐	☐	☐	☐
Consistent usage of language	☐	☐	☐	☐	☐	☐
Requirement is too long	☐	☐	☐	☐	☐	☐
Adherence to templates	☐	☐	☐	☐	☐	☐
Validation	☐	☐	☐	☐	☐	☐
Testability of requirements	☐	☐	☐	☐	☐	☐
Quality scoring system	☐	☐	☐	☐	☐	☐

Duplicated requirements in the same specification	☐	☐	☐	☐	☐	☐
Duplicated requirements in any specification	☐	☐	☐	☐	☐	☐

Page 09

17. Do you have any additional comments, questions, or concerns you that would like to address?

Clicking 'Next' will submit your answers.

Last Page

Thank you for completing this survey!

We would like to thank you very much for helping us.

Your answers were transmitted, you may close the browser window or tab now.